Università degli Studi di Padova

Head Office: Università degli Studi di Padova

Department of Mathematics
Ph.D. Course in: Brain, Mind and Computer Science
Curriculum: Computer Science for societal challenges and innovation
Series: XXX

Definition and learning of logic-based kernels for categorical data, and application to collaborative filtering

Supervisor: Prof. Fabio Aiolli
Co-Supervisor: Prof. Anna Spagnolli

Ph.D. student: Mirko Polato

Abstract

The continuous pursuit of better prediction quality has gradually led to the development of increasingly complex machine learning models, e.g., deep neural networks. Despite the great success in many domains, the black-box nature of these models makes them not suitable for applications in which the model understanding is at least as important as the prediction accuracy, such as medical applications. On the other hand, more interpretable models, as decision trees, are in general much less accurate. In this thesis, we try to merge the positive aspects of these two realities, by injecting interpretable elements inside complex methods. We focus on kernel methods which have an elegant framework that decouples learning algorithms from data representations.

In particular, the first main contribution of this thesis is the proposal of a new family of Boolean kernels, i.e., kernels defined on binary data, with the aim of creating interpretable feature spaces. Assuming binary input vectors, the core idea is to build embedding spaces in which the dimensions represent logical formulas (of a specific form) of the input variables. As a result the solution of a kernel machine can be represented as a weighted sum of logical propositions, and this allows to extract from it human-readable rules. Our framework provides a constructive and efficient way to calculate Boolean kernels of different forms (e.g., disjunctive, conjunctive, DNF, CNF). We show that on binary classification tasks over categorical datasets the proposed kernels achieve state-of-the-art performances. We also provide some theoretical properties about the expressiveness of such kernels.

The second main contribution consists in the development of a new multiple kernel learning algorithm to automatically learn the best representation (avoiding the validation). We start from a theoretical result which states that, under mild conditions, any dot-product kernel can be seen as a linear non-negative combination of Boolean conjunctive kernels. Then, from this combination, our MKL algorithm learns non-parametrically the best combination of the conjunctive kernels. This algorithm is designed to optimize the radius-margin ratio of the combined kernel, which has been demonstrated of being an upper bound of the Leave-One-Out error. An extensive empirical evaluation, on several binary classification tasks, shows how our MKL technique is able to outperform state-of-the-art MKL approaches.

A third contribution is the proposal of another kernel family for binary input data, which aims to overcome the limitations of the Boolean kernels. In this case the focus is not exclusively on the interpretability, but also on the expressivity. With this new

framework, that we dubbed propositional kernel framework, is possible to build kernel functions able to create feature spaces containing almost any kind of logical propositions.

Finally, the last contribution is the application of the Boolean kernels to Recommender Systems, specifically, on top-N recommendation tasks. First of all, we propose a novel kernel-based collaborative filtering method and we apply on top of it our Boolean kernels. Empirical results on several collaborative filtering datasets show how less expressive kernels can alleviate the sparsity issue, which is peculiar in this kind of applications.

Acknowledgements

Incommensurable appreciation and gratitude for the help and support is extended to the following persons who have, in different ways, contributed to the success of my graduate studies.

Foremost, I wish to thank my advisor Prof. Fabio Aiolli for his guidance and support in both research and life in general. His teachings and valuable advices have helped me to become a better researcher. Besides my advisor, I would like to thank my family for giving me the opportunity to follow my dreams and to realize my own potential.

I am also thankful to Prof. Alessandro Sperduti who drove me towards this Ph.D. adventure by supporting me during the Master degree and in the early stages of my doctoral. Next, I thank Prof. Martha Larson for giving me the opportunity to visit the Multimedia Computing group at Delft University of Technology.

Moreover, I wish to thank my friends who helped me to get through the tough times. Finally, thanks to all the people that supported me during these years.

Contents

IV Conclusions

PART

I

Introduction & Background

Summary

1

Introduction

Man's reach exceeds his imagination.

The Prestige
Robert Angier

In short

This chapter opens the curtain of this thesis by introducing the motivations and the outline of the main contributions.

1.1 Motivations

The impressive growth of available data in the last years has offered fertile ground for machine learning methods to show their capabilities. Machine learning is the branch of Artificial Intelligence that deal with the development of algorithms able to learn from experience [Mit97, Alp10]. Given a specific task, and data which represent the past experience, learning (for a machine) means optimizing some set of parameters of a model by exploiting the data. Such learned model should be able to make predictions or to infer useful knowledge from data.

Machine learning can be seen as a point of conjunction between mathematics, statistics, and computer science. Mathematics and statistics offer the instruments for creating theoretically well founded models, while computer science provides the tools for implementing such models inside the machine.

However, machine learning is not only a matter of good models and efficient code, it is also, and maybe most, a matter of appropriate data representation. Machines, in their lower level, can understand only zeros and ones, while data can have the most disparate

forms, and finding an effective way to represent the available knowledge is not trivial.

1.1.1 The representation problem

The representation problem refers to the problem of converting the available data into a representation which ideally distill all the relevant information about the task in a compact manner.

Considering that data are the input of the model, this problem of finding how to describe the available knowledge is faced since the early stages of a learning process. Typically, this can be considered as a pre-training step in which the best set of features, concerning the learning task, are, implicitly or explicitly, defined. In the past, the most used approach was to fix an a-priori representation, usually extracted from a "manually" performed feature engineering step [HS05]. Thus, the efforts were mainly devoted to the search of an appropriate function to solve the given task. Recently, a new paradigm which involves the learning of the best representation is continuously gaining interest from the research community. The most representative example of this trend is deep learning [BCV13, Sch15]. Deep learning models are composed by a sequence of layers which encode the data representation with different levels of abstraction. In this way, the burden of creating the best representation is entrusted to the algorithm itself. Deep neural networks, and all the different kinds of recurrent networks, have improved the state-of-the-art in many domains, especially the ones regarding multimedia data [BCS+16, RDGF16, GSK+17, DHR+17].

However, the biggest shortcoming of deep models is their complex structure which makes them sort of black-boxes, despite there have been some attempts to extract interpretable rules from them [ZLMJ16, BAJB17, Tsu00, EL06, GT10]. Moreover, other issues related to the standard neural networks need to be mentioned [LBLL09, EMB+09, KK01, PMB13, SMDH13]:

- by design, they do not provide any decoupling between the representation and the model;
- they are demanding, especially in deep architectures, in terms of resources and training time;
- they are theoretically not well understood;
- the convergence to the optimal solution is not guaranteed because of the presence of local minima and the gradient vanishing phenomenon.

An alternative to the deep approach, is represented by kernel methods, which have had a huge impact in the research community in the first years of the new millennium [SS01, STC04, HSS08, STV04]. Nonetheless, they are still used in many learning

tasks since they are generally less demanding than deep networks. Kernel methods, as the name suggest, exploit a very solid theoretical framework concerning kernels.

Loosely speaking, a kernel is a function which represents a similarity in some vector space. This vector space is induced by a mapping function which takes the input vector and projects it onto an high dimensional space. What makes such kernel functions so appealing is that this high dimensional mapping is implicitly performed, and hence its computation is possible and efficient. Thus, defining a kernel function, and its corresponding mapping, means defining a new data representation. This allows the application of kernel methods to different kinds of data, as long as a suitable kernel function is provided. Furthermore, kernel methods offer an elegant and theoretically founded framework that decouples learning algorithms from data representation. Given a dataset and a kernel method, it is possible to apply on it different kernels without changing the behaviour of the method, which simply solves the problem inside the new space defined by the kernel.

In the direction of learning the representation instead of fixing it a-priori, kernel learning has emerged as a new research topic. Given a specific task or set of tasks, in kernel learning the goal is to learn the optimal kernel (i.e., the optimal representation), and the corresponding implicit feature map. Multiple Kernel Learning (MKL) [DKWH09, GA11, ADNS15, AD15] is one of the most successful approaches to kernel learning. MKL methods are designed to combine a set of base (a.k.a. weak) kernels to obtain a better one. Nevertheless, also kernel methods have their own weaknesses:

- conversely to the deep networks, the representation offered by a kernel is shallow;
- they suffer from scalability issues, since they usually required to store in memory the full kernel matrix which may become huge when the number of examples is very large;

Finally, even though much less than deep networks, kernel methods are also treated as black-box models, and hence their interpretation is not straightforward. However, in the literature there are approaches that try to extract rules from kernel methods (specifically SVM) [GT10, BB10, ZSJC05, Die08, FSR05].

1.1.2 Interpretability

The lack of interpretability of many machine learning algorithms makes hard their application in scenarios in which explanations are as important (if not more) as the prediction quality. For example, in biological/bioinformatic applications it is not always enough to have a model able to predict whether or not a specific disease is going to exhibit. Instead, it would be very helpful to understand which are the bio-markers (i.e., features) related to that specific disease in order to give to the practitioners useful information to work with.

Regarding the need of explanations, in 2016, the European Parliament adopted a set of comprehensive regulations for the collection, storage and use of personal information, the General Data Protection Regulation [GF16]. Inside such regulation, that slates to take effect as law across the EU in 2018, the first line of the Article 22.1 states that: "*The data subject shall have the right not to be subject to a decision based solely on automated processing*", and later in the same article it is specified that such decisions must be supervised by human beings. In these contexts, having the possibility of knowing which are the key elements that guide the machine's decision can be helpful for the human supervision.

Beyond these delicate scenarios, another example in which explanations play a huge role are recommender systems. When a recommendation is conveyed to the user, it can be accepted or rejected. In case of rejection, without any explanation about the reason why such recommendation has been provided, the user can lose the trust on the system, because it simply does not work as expected. On the contrary, with the help of a reasonable explanation, a bad recommendation can be "justified" by the user [PC07]. Moreover, explanations can also play an active role in the recommendation: anytime a user does not know anything about the recommended item, the explanation can provide useful insights about why such item can be of interest for him/her. For these reasons, recently, this issue is gaining attention from the research community [Tin07, CMEC17, HVPD17].

The ability to understand machine learning models, is not only useful for the end user, but it is also important for researchers since it may help in further understanding the models and in designing better ones.

1.2 Contributions

The aim of this thesis is to provide novel methods for facing both the representation and the interpretability problem. In particular, these problems are addressed in contexts in which the input data is binary and/or categorical.

The original contributions of this thesis are listed in the following:

- Proposal of a **new family of kernels for binary data** which creates easy-to-interpret feature spaces. The core idea of this family of (Boolean) kernels is the definition of embedding spaces in which the dimensions represent logical formulas over the input variables. This framework provides a constructive and efficient way to calculate many kernels which are able to mimic logical operations, such as disjunctions, conjunctions, DNFs and CNFs. Empirical analysis show that, on binary classification tasks, the proposed kernels achieve state-of-the-art performance on several benchmark categorical datasets;

- **Analysis of the expressiveness** of the Boolean kernels and the relation with their capability of dealing with sparse data. This analysis is performed from both the theoretical and the empirical point of view;
- **Theoretical connection** between Boolean kernels and dot-product kernels. We show how dot-product kernels can be computed as linear non-negative combinations of monotone conjunctive Boolean kernels;
- Proof of concept algorithm, based on a genetic approach, for interpreting the solution of a SVM. The application on a set of artificial and benchmark datasets shows the potential of Boolean kernels for extracting human-readable rules that explain the decision;
- Proposal of **a novel multiple kernel learning algorithm**, dubbed GRAM, designed to optimize the radius-margin ratio, which has been demonstrated of being an upper bound of the Leave-one-out error. Empirical results on several benchmark categorical datasets show that GRAM is able to outperform state-of-the-art MKL methods;
- Proposal of **a second kernel family** for binary data, called propositional kernels, which aspires to overcome the limitations of the Boolean kernels. The aim of the propositional kernels is not exclusively focused on the interpretability, but also on the expressive power. This framework is able to build feature spaces, and corresponding kernel functions, that can potentially contain almost any kind of logical propositions. The framework provides all the theoretical tools for constructing such kernels in a constructive and efficient manner. An example of application, on binary classification tasks, shows the potential of this family of kernels;
- **Application** of kernel methods based on Boolean kernels **on Recommender Systems**, specifically, on top-N recommendation. This contribution aims to underline the applicability of the Boolean kernel framework in a real case-study where explanations can be helpful. Firstly, an efficient kernel-based collaborative filtering method is proposed. Then, it is applied on several collaborative filtering datasets. Results show that Boolean kernels are able to alleviate the sparsity issue, which is peculiar in this kind of applications. Moreover, the resulting decision function can be used to extract explanation rules for the users, for example, by using the method proposed in Chapter 7.

1.3 Publications

Most of the content of this thesis has been presented in international peer-reviewed conferences and journals. The complete list of already published papers is shown in the following.

Journals

J01 **Exploiting sparsity to build efficient kernel based collaborative filtering for top-N item recommendation**. Mirko Polato and Fabio Aiolli. Neurocomputing, vol. 268, pp. 17-26, 2017.

Conferences

C01 **Kernel based collaborative filtering for very large scale top-N item recommendation**. Mirko Polato and Fabio Aiolli. In Proceedings of the 25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning', ESANN '16, Bruges (Belgium) 2016.

C02 **Classification of categorical data in the feature space of monotone DNFs**. Mirko Polato, Ivano Lauriola and Fabio Aiolli. In Proceedings of the 26th International Conference on Artificial Neural Networks, ICANN '17, Alghero (Italy), 2017.

C03 **Radius-margin ratio optimization for dot-product boolean kernel learning**. Ivano Lauriola, Mirko Polato and Fabio Aiolli. In Proceedings of the 26th International Conference on Artificial Neural Networks, ICANN '17, Alghero (Italy), 2017. Artificial Intelligence and Statistics, AISTATS '18, 2018.

Workshops

W01 **A preliminary study on a recommender system for the job recommendation challenge**. Mirko Polato and Fabio Aiolli. In Proceedings of the Recommender Systems (RecSys) Challenge 2016, Boston (USA, Massachusetts), 2016.

W02 **Disjunctive Boolean Kernels based Collaborative Filtering for top-N item recommendation**. Mirko Polato and Fabio Aiolli. In Proceedings of the 8th Italian Information Retrieval Workshop, IIR '17, Lugano (Svizzera), 2017.

1.4 Document structure

The thesis is divided in four main parts:

PART I, which has started with this chapter, provides all the background knowledge necessary to fully understand the remainder of the thesis. We start by defining the notations used throughout the document, and we also give the descriptions of the datasets and the metrics used in our experiments. Then, Chapter 4 firstly presents the learning problem in general, and in the successive sections it goes a bit deeper in the description of kernel methods and all the other concepts that will be useful for the understanding of this thesis.

PART II describes the first main contribution, namely the Boolean kernel framework. In Chapter 6 the Boolean kernel family is presented. For each kernel the description and the method to compute it is provided, as well as a discussion of its properties. In the last part of the chapter the evaluation assessment of these kernels is presented and discussed. Chapter 7 presents a proof of concept application of the Boolean kernels for interpreting the solution of a kernel machine. Chapter 8 describes the second main contribution, that is the propositional kernel framework: first of all, the weaknesses of the Boolean kernels are highlighted, and then a description of the propositional kernel framework is provided. Finally, an example of application of the framework on binary classification tasks is presented. Lastly, the third main contribution is presented in Chapter 9. Our MKL algorithm, dubbed GRAM, is described from both a theoretical and the algorithmic point of view. Finally, a thorough evaluation against other state-of-the-art MKL approaches is performed.

PART III presents the application of the Boolean kernels to the top-N recommendation problem. We first propose a new kernel-based collaborative filtering method for implicit feedback datasets. Then, an analysis on how the sparsity influences the kernels is provided. Finally, an extensive evaluation on several collaborative filtering datasets is discussed.

PARTI IV concludes the thesis: Chapter 11 wraps up some considerations about this dissertation and the achieved results of our research. Finally, it gives some possible research paths that can be followed in the future in order to give continuity to this research.

2

Notation

> We could, of course, use any notation we want; do not laugh at notations; invent them, they are powerful. In fact, mathematics is, to a large extent, invention of better notations.
>
> *The Feynman Lectures on Physics, Addison-Wesley, Chapter 17 (1963)*
> Richard Phillips Feynman

In short

This chapter introduces the main notations used throughout this thesis. Some of the concepts treated in this chapter will be resumed in the remainder for a more thorough explanation, however, they are introduced here only to contextualize the notation. Table 2.1 presents the fundamental symbols exploited in the following. Notation regarding binary classification and recommender systems are presented in Section 2.1 and Section 2.2, respectively.

2.1 Binary classification

A generic supervised learning task consists in inferring a target function from labeled data (i.e., examples). In the context of supervised binary classification, examples are a set of pairs of the form $\{(\mathbf{x}_i, y_i)\}_{i=1}^{L}$, with vectors $\mathbf{x}_i$ in some set $\mathcal{X}$ and labels $y_i \in \{-1, +1\}$. Usually, in order to assess the goodness of a learning algorithm, the whole set of data is divided into two sets: a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ used to build the model, and a test set $\{(\mathbf{x}_i, y_i)\}_{i=l+1}^{L}$ used to estimate the algorithm's performance. In this document, training-and-test division is always assumed.
When it is not specified differently the dimension of the vectors $\mathbf{x}_i$ is assumed to be n.

Symbol	Interpretation
$\mathbf{v}$	Bold-faced lower case letters are used to identify vectors, which are considered by default column vectors
$\mathbf{v}_i$	The indexed notation identifies the i-th element of the vector $\mathbf{v}$
$\mathbf{1}_n$	A vector in $\mathbb{R}^n$ with all entries set to 1
$\langle \cdot, \cdot \rangle$	The dot-product (a.k.a. inner or scalar product) of vectors
$\mathbf{M}$	Bold-faced capital letters are used to identify matrices
$\mathbf{M}_{i,j}$	The indexed notation identifies the j-th element in the i-th row of the matrix $\mathbf{M}$
$\mathbf{M}_{i,:}$	The i-th row vector of the matrix $\mathbf{M}$
$\mathbf{M}_{:,j}$	The j-th column vector of the matrix $\mathbf{M}$
$\mathbf{1}_{n \times m}$	A vector in $\mathbb{R}^{n \times m}$ with all entries set to 1
$\mathbf{I}_n$	The identity matrix in $\mathbb{R}^{n \times n}$
$\odot$	The element-wise product of vectors or matrices
$\lVert \cdot \rVert_1$	The L^1 norm (Taxicab norm) of a vector or a matrix
$\lVert \cdot \rVert_2$	The L^2 norm of a vector or a matrix
$\lVert \cdot \rVert_{\mathrm{F}}$	The Frobenius norm of a matrix
$\lVert \cdot \rVert_{\mathrm{T}}$	The trace norm (nuclear norm) of a matrix
$\cdot^{\intercal}$	The transpose of a vector or a matrix
$\mathcal{O}$	The big O notation
$\lvert \mathcal{A} \rvert$	The cardinality (i.e., number of elements) of the set $\mathcal{A}$
$[n]$	The set of all integers between 1 and n, i.e., $\{i \in \mathbb{Z}^+ \mid i \leq n\}$
$[\![P]\!]$	The indicator function which returns 1 if the predicate P is true, and 0 otherwise

Table 2.1: Summary of the used notation throughout this thesis.

2.2 Recommender systems

Broadly speaking, Recommender Systems (RSs) are information filtering technologies used to provide personalized advices. The set of those who receive suggestions from the system are usually called users, and they are indicated by the set $\mathcal{U}$, with $|\mathcal{U}| = n$. While, what the system suggests to users are called items, and they are denoted by the set $\mathcal{I}$, with $|\mathcal{I}| = m$.

In general, in order to give personalized recommendations, RSs leverage on historical data about users' interactions with items. These interactions, also referred to as feedbacks, can be divided into implicit and explicit feedbacks. Implicit feedbacks are those user-item interactions which come from a natural use of the system. In particular, the user is not asked to give any kind of direct preference to the items. Examples of these kind of interactions are clicks, views, time spent on a webpage and so on. Conversely, explicit feedbacks are interactions which represent unambiguously the user's taste. The most common example of explicit feedback is the rating, e.g., 1-to-5 stars and thumbs up versus thumbs down.

In this document, implicit feedbacks are considered as binary information about interactions between users and items (i.e., exists/not exists an interaction), and they are collected in the so-called binary rating matrix $\mathbf{R} \in \{0,1\}^{n \times m}$, where users are row vectors and items are column vectors. By leveraging on the analogy between binary vectors and sets, the matrix $\mathbf{R}$ will be sometimes indicated by its corresponding set of binary ratings $\mathcal{R} \equiv \{(u,i) \mid u \in \mathcal{U}, i \in \mathcal{I}, \mathbf{R}_{u,i} = 1\}$.

> **Slight abuse of notation**
>
> The notation $\mathbf{R}_{u,i}$ requires that $\mathcal{U} \equiv [n]$ and $\mathcal{I} \equiv [m]$, which may not be always the case. In order to ease the readability, the reader can assume that for any set $\mathcal{A}$ exists a bijective function $f : \mathcal{A} \to \mathbb{Z}^+$ which maps each element of $\mathcal{A}$ onto an unique integer in the interval $[|\mathcal{A}|]$.

Similarly to the ratings case, the set of items rated by a users u and the set of users who rated the item i are indicated by the sets $\mathcal{I}_u \equiv \{i \in \mathcal{I} \mid (u,i) \in \mathcal{R}\}$, and $\mathcal{U}_i \equiv \{u \in \mathcal{U} \mid (u,i) \in \mathcal{R}\}$, respectively.

3

Evaluation

> Not everything that can be counted counts, and not everything that counts can be counted.
>
> Albert Einstein

In short

In this chapter we present the metrics and the datasets used during the empirical evaluations. Firstly, we describe metrics for both the binary classification task and the recommendation task and then we give a brief description of the benchmark datasets.

3.1 Metrics

3.1.1 Binary classification

Choosing the appropriate evaluation metric is very important in order to find which model achieves the best performance. Many measures for evaluating classification (and information retrieval in general) models have been proposed, but certainly the most widely used are: precision, recall, accuracy and AUC.

Given a binary classifier and an instance, there are four possible outcomes:

True Positive (TP) the instance is positive (p) and it is classified as positive (p');

False Negative (FN) the instance is positive (p) and it is classified as negative (n');

True Negative (TN) the instance is negative (n) and it is classified as negative (n');

False Positive (FP) the instance is negative (n) and it is classified as positive (p').

The confusion matrix presented in Figure 3.1 summarizes these possible scenarios.

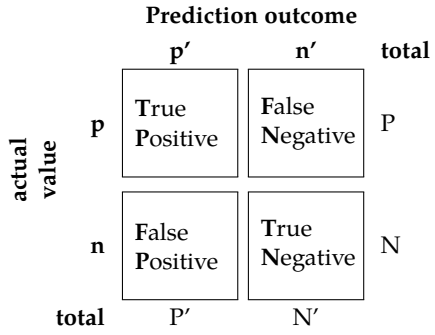


Figure 3.1: Confusion matrix for binary classification.

Precision, recall and accuracy

The accuracy is often the starting point for analyzing the quality of a classification model. Accuracy measures the ratio of correct classifications (i.e., TP and TN) to the total number of cases evaluated. The analytic formula is the following:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.1}$$

Despite its popularity, the accuracy suffers of a critical drawback, known as *accuracy paradox*: in the case of unbalanced datasets, a classifier that predicts the dominant (i.e., the most probable) class could achieve a very good accuracy. For this reason, accuracy is usually reported along with precision and recall.

Precision is calculated as the number of TP divided by the total number of elements labeled as positives by the classifier, i.e., TP and FP. Recall (also known as *sensitivity* or *true positive rate*) is defined as the number of TP divided by the total number of elements that are actually positives, i.e., TP and FN.

Formally:

$$\text{precision} = \frac{\text{TP}}{\text{P}'} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{recall} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3.2}$$

AUC

Besides the true positive rate (TPR - i.e., recall), it is also useful to define the *false positive rate* (FPR), as the ratio between negative examples classified as positives and the total number of negative examples:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{3.3}$$

The higher the TPR the better is the classifier, and conversely, the lower the FPR the better is the classifier. This tradeoff, usually called benefits (TP) and costs (FP) tradeoff, can be summarized using the Receiver Operating Characteristics (ROC) graph. ROC graphs are two-dimensional graphs in which TPR is plotted on the y axis and FPR is plotted on the x axis. Loosely speaking, one point (i.e., a classifier performance) in ROC space is better than another if it is on the upper-left of the latter.

A single point in the ROC space represents the goodness of a discrete binary classifier. However, anytime a classifier instead of returning directly a predicted class label returns a score (e.g., the probability of being positive/negative), its output can be interpreted as a ranking. Such ranking can be used with a threshold to produce a discrete binary classification, e.g., a score above the threshold means positive, and negative otherwise. Each threshold value produces a potential new point in the ROC space, and ideally by varying the threshold inside the interval $(-\infty, +\infty)$ it is possible to trace a curve which starts in the lower left corner (i.e., TPR=FPR=0) and ends up in the upper right corner (i.e., TPR=FPR=1).

The ROC curve unbinds the assessment of the quality of a ranker from the choice of a specific classification threshold. A possible way to compare two ROC curves is by comparing the underlying area: the greater the area the better the performance of the classifier. This metric is known as Area Under the ROC Curve, or simply AUC (or AUROC) [Faw06]. From a computational point of view, the definition of the AUC given above is not the most practical one. However, it is also possible to see the AUC from another perspective, that is, it represents the probability that the ranker will rank a randomly chosen positive instance higher than a randomly chosen negative instance (assuming positive ranks higher than negative). This is equivalent to the Wilcoxon test of ranks [HM82]. This new view of the AUC allows to give a reasonably efficient way to calculate it: given the predicted scores of n instances, $\mathbf{s} \in \mathbb{R}^n$, then

$$\text{AUC}(\mathbf{s}) = \frac{1}{n_\oplus n_\ominus} \sum_{i=1}^{n} \sum_{j=1}^{n} [\![\mathbf{s}_i > \mathbf{s}_j]\!] \in [0, 1], \tag{3.4}$$

where $n_\oplus$ and $n_\ominus$ represent the number of positive and negative examples, respectively.

3.1.2 Top-N recommendation

In top-N recommendation the quality of a method depends on how good the produced items ranking is for a user, or in other words, the more relevant items are in the top of the ranking, the better is the method. A ranking metric has already been presented in Section 3.1.1, namely the AUC, and it applies also for top-N recommendation. Besides this full-rank metric, many other metrics are very popular in the RSs community and most of them try to reward the rankings which have many relevant items at the very

top. Some of them are described in the following.

Average Precision

When additional emphasis on the correctness of the few top-ranked items is required, the Average Precision (AP) metric is preferred to the AUC [SYZ15]. As for the AUC, AP can be seen as an area under a curve, specifically the precision-recall curve: given a ranking $\mathbf{s}_u$ over n items for a user u, this area (i.e., AP) can be approximated by the following finite sum:

$$\text{AP}(\mathbf{s}_u) = \frac{1}{|\mathcal{T}_u|} \sum_{k=1}^{n} [\![\mathbf{s}_{uk} \in \mathcal{T}_u]\!] \cdot \text{precision@}k(\mathbf{s}_u), \tag{3.5}$$

where $\mathcal{T}_u$ is the set of positive (i.e., relevant) items for the user u in the test set, and $\text{precision@}k(\mathbf{s}_u)$ indicates the precision up to the k-th element in the ranking, and it is calculated by

$$\text{precision@}k(\mathbf{s}_u) = \frac{1}{k} \sum_{i=1}^{k} [\![\mathbf{s}_{ui} \in \mathcal{T}_u]\!]. \tag{3.6}$$

Usually, in order to accentuate the importance of the top of the ranking, the AP is limited to a certain position k, as done for the precision, and it is referred to as AP@k. The average over all the users' AP@k is called Mean Average Precision at k (mAP@k), and it is computed as follows:

$$\text{mAP@}k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \text{AP@}k(\mathbf{s}_u). \tag{3.7}$$

3.2 Datasets

In this section, we present the datasets used as benchmark for both binary classification and recommendation. For each dataset we provide a brief description, the source where it is possible to get it and the information about the typology, the number of features, and the number of examples.

3.2.1 Binary classification

Since our models require binary input vectors, we selected datasets with binary or categorical features in such a way that the binarization process do not loose information. In particular, for each dataset the following preprocessing steps have been performed:

- instances with missing attributes have been removed;

- categorical features, including the binary ones, have been mapped into binary features by means of the *one-hot* encoding [HH13]. This preprocessing keeps for every example in the dataset the same number of ones (m), or in other words every input vector has the same L^1-norm;
- non binary tasks (`primary-tumor`, `soybean` and `dna`) have been artificially transformed into binary ones, by arranging the classes into two groups while trying to keep the number of instances balanced.

In the following a brief description of the benchmark datasets is provided:

`dna` This dataset contains DNA sequences.

`house-votes` This dataset includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac.

`kr-vs-kp` Series of final positions in a chess game: in every game white has the king and a rook, while black has the king and a pawn on a2 (one square away from promotion). The task is to predict whether white can win or not.

`monks` This is a set of three artificial datasets (**`monks-1`**,**`monks-2`** and **`monks-3`**) over the same attributes space. It has been created to test a wide range of induction algorithms as reported in [TBB+91].

`primary-tumor` This is one of three domains provided by the Ljubljana Oncology Institute that has repeatedly appeared in the machine learning literature.

`promoters` This dataset contains Escherichia Coli promoter gene sequences (DNA), and it has been developed to help evaluate a learning algorithm called KBANN [TSN90].

`soybean` The dataset contains examples of soybean with 35 nominal attributes that describe the status of the soybean. The task is the diagnosis of soybean's disease.

`spect` The dataset describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patient is classified into two categories: normal and abnormal.

`splice` Splice junctions are points on a DNA sequence at which superfluous DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (parts of the DNA sequence retained after splicing) and introns (parts of the DNA sequence that are spliced out).

Dataset	#Instances	#Features	Distribution(%)	$m = \|\mathbf{x}\|_1$
dna	2000	180	47/53	47
house-votes	232	32	47/53	16
kr-vs-kp	3196	73	52/48	36
monks-1	432	17	50/50	6
monks-2	432	17	33/67	6
monks-3	432	17	53/47	6
primary-tumor	339	34	42/58	15
promoters	106	228	50/50	57
soybean	266	97	55/45	35
spect	267	45	79/21	22
splice	3175	240	48/52	60
tic-tac-toe	958	27	65/35	9

Table 3.1: Datasets information: name, number of instances, number of features, classes distribution and number of active variables in every example (i.e., $\|\mathbf{x}\|_1$).

`tic-tac-toe` This dataset encodes the complete set of possible board configurations at the end of tic-tac-toe games, where $\times$ is assumed to have played first. The target concept is "win for $\times$".

Table 3.1 and Table 3.2 summary the information regarding the above described datasets.

Dataset	Reference	URL
dna	[HL02]	[2]#dna
house-votes	[Sch87]	[1] /congressional+voting+records
kr-vs-kp	[Sha87]	[2]/Chess+(King-Rook+vs.+King-Pawn)
monks	[TBB+91]	[2]/MONK's+Problems
primary-tumor	-	[2]/primary+tumor
promoters	[TSN90]	[2]/Molecular+Biology+(Promoter+Gene+Sequences)
soybean	[MC80]	[2]/Soybean+(Large)
spect	[KCT+01]	[2]/spect+heart
splice	[NTS90]	[2]/Molecular+Biology+(Splice-junction+Gene+Sequences)
tic-tac-toe	[MR89]	[2]/Tic-Tac-Toe+Endgame

Table 3.2: Useful references about the datasets.

3.2.2 Recommender Systems

To assess the quality of the top-N recommendation we used many benchmark datasets available online. Since we assume to have only implicit feedback, for each non-binary dataset we considered a rating as an implicit feedback regardless the value of the rating. For the **Jester** dataset since it is almost fully dense, we considered only the positive ratings as implicit feedbacks.
In the following a brief description of the benchmark datasets for the recommendation is provided:

BookCrossing Collected by Cai-Nicolas Ziegler in a 4-week crawl in 2004 from the BookCrossing community. The full version contains roughly 30k users (anonymized but with demographic information) providing over one million ratings (explicit / implicit) about books.

Ciao Ciao (DVD) is a dataset crawled from the entire category of DVDs from the dvd.ciao.co.uk website in December 2013.

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html
[2] https://archive.ics.uci.edu/ml/datasets/

Dataset	Item type	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{R}\rvert$	Density
`BookCrossing` [3]	Books	2802	5892	70593	0.004%
`Ciao`	Movies	17615	16121	72664	0.025%
`FilmTrust`	Movies	1508	2071	35496	1.13%
`Jester` [4]	Jokes	24430	100	1M	42.24%
`MovieLens`	Movies	6040	3706	1M	1.34%
`MSD`	Music	1.2M	380K	48M	0.01%
`Netflix` [3]	Movies	93705	3561	3.3M	0.99%

Table 3.3: Datasets information: name, type of the items, number of users, number of items, number of ratings and density of the ratings.

`FilmTrust` FilmTrust is a small dataset crawled from the entire FilmTrust website in June 2011.

`Jester` Anonymous ratings data from the Jester Online Joke Recommender System. Ratings are in a continuous scale between -10 and +10.

`MovieLens` Dataset collected by the GroupLens Research Project at the University of Minnesota. The dataset consists of 5 stars ratings about movies. Each user has rated at least 20 movies. It also contains simple demographic info for the users (e.g., age, gender, occupation, zip). We used the version with 1M ratings.

`MSD` The Million Song Dataset (MSD) is a freely-available collection of audio features and metadata for a million contemporary popular music tracks.

`Netflix` Dataset from the Netflix Prize competition held by Netflix from 2007 to 2009. It contains explicit ratings about movies, and also some information about the movies.

Table 3.3 and Table 3.4 summarize the information regarding the above described datasets.

[3] We used a reduced version of the full dataset.

[4] For experimental purposes, we removed from the `Jester` dataset all the users with more than 90 ratings (i.e., $> 90\%$ of the items).

Dataset	Reference	URL
BookCrossing	[ZMKL05]	http://grouplens.org/datasets/book-crossing
Ciao	[GZTYS14]	http://www.librec.net/datasets.html#ciaodvd
FilmTrust	[GZYS13]	http://www.librec.net/datasets.html#filmtrust
Jester	[GRGP01]	http://goldberg.berkeley.edu/jester-data/
MovieLens	[HK15]	http://grouplens.org/datasets/movielens
MSD	[MBMEL12]	https://labrosa.ee.columbia.edu/millionsong/
Netflix	[BL]	http://www.netflixprize.com

Table 3.4: Useful references about the datasets.

4

Learning with kernels

> The key to artificial intelligence has always been the representation.
>
> Jeff Hawkins

In short

In the opening lines of the preface of his textbook *Machine Learning* [Mit97], Tom Mitchell provides the following informal definition of machine learning:

> *The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.*

Afterwards, in the introduction, he gives a more rigorous definition:

Definition : Machine Learning [Mit97]

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Even though it seems rather formal, this definition encloses all the fundamental ingredients of any machine learning method. Let us break down this definition by instantiating each formal concept to something more concrete. In a single sentence, we can say that in machine learning problems we try to discover patterns from data [SS01].

What Mitchell calls experience E is indeed data, which represents the most important element of all. Without data there would be no learning, as for a human being without experience. In life, we learn to perform a specific task whenever our experience on that task is enough to master it. Same for machines, they learn to perform a task T from data that contain useful information for T. Different tasks can require different kind of information. Finally, in order to assess whether there is any learning, we need some way to measure the improvements (i.e., the performance measure P). For human beings, this P is usually a qualitative assessment rather than a quantitative one, however, for machines, in most of the cases, we need to be as objective as possible and so the performance must be quantifiable.

Starting from this general definition of machine learning (ML), in the following of this chapter we will go a bit deeper inside one of the most successful family of ML approaches, namely kernel methods. We will also try to give all the necessary background to fully understand the remainder of this thesis.

4.1 Learning and data representation

Data represents the experience for the machine. With data we refer to any piece of information regarding a particular aspect of the reality (assuming artificially generated data as a special kind of reality). However, this is a rather fuzzy definition, and for our purposes we need something more specific, but at the same time as much general as possible. A machine learning problem consists in finding the relation that binds input objects to specific target values. The standard way to formalize this concept is as follows: we are given the empirical data (or *training data*)

$$\mathcal{D} \equiv \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots (\mathbf{x}_l, y_l)\}, \quad \forall\, i \in [l],\ (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}. \tag{4.1}$$

Here, $\mathcal{X}$ is some nonempty set with elements $\mathbf{x}_i$ usually called *instances*, *patterns* or *inputs*. The set $\mathcal{Y}$ contains all the possible *targets* (a.k.a. *labels* or *outputs*) and, differently from $\mathcal{X}$, it can be empty: in that case there are no targets associated with the patterns, and the learning task is said to be *unsupervised*. This terms comes from the fact that the supervision (i.e., the targets) is missing, and all we can do is finding relations between the patterns themselves. Conversely, anytime $\mathcal{Y}$ is nonempty the task is called *supervised*.

By looking at the set $\mathcal{Y}$ we can also distinguish other two classes of learning tasks, namely *classification* (a.k.a. *pattern recognition*) and *regression*. As the name suggests, a classification task consists in correlating patterns to their corresponding classes, and generally the number of classes is limited and always finite, i.e., $|\mathcal{Y}| < \infty$. On the other hand, when the targets can potentially assume an infinite range of values (e.g., real-valued outputs) the learning task is called regression.

Loosely speaking, the goal of (supervised) learning is to discover which relation links the inputs to the outputs. Formally, given the empirical data $\mathcal{D}$, in which we assume exists a function (i.e., relation) $f : \mathcal{X} \rightarrow \mathcal{Y}$ such that $\forall\ (\mathbf{x}_i, y_i) \in \mathcal{D}, f(\mathbf{x}_i) = y_i$, a learning algorithm tries to find the function (a.k.a. model or hypothesis) h that approximates f as tightly as possible.

Assumption on the input set $\mathcal{X}$

It is noteworthy that we did not make any assumption on the set $\mathcal{X}$ which can contain any kind of object, e.g., cars, apples or graphs. However, in order to ease the reading, and for being consistent with the notation, throughout the document we assume that $\mathcal{X} \subseteq \mathbb{R}^n$, for some $n \in \mathbb{N}^+$.

A crucial aspect of learning is that the available empirical data (or simply data), typically, do not cover every possible facets of the reality, and hence we have to be able to *generalize* in order to respond to "similar" stimuli in "similar" ways. In learning, both notions of generalization and similarity are essential. In order to generalize to unseen instances, we have to leverage on our experience (i.e., $\mathcal{D}$): we want to associate to a new input $\mathbf{x}_{\text{new}}$ an output $y \in \mathcal{Y}$ which is related with an already seen pattern $\mathbf{x}$ (i.e., $(\mathbf{x}, y) \in \mathcal{D}$) that is similar to $\mathbf{x}_{\text{new}}$, for some notion of similarity.

The choice of the similarity measure is one of the core questions in ML, and it is strictly related to the representation (of the inputs) problem. The representation problem can be divided into two sub-problems: (i) which are the relevant information to consider; (ii) how to represent such knowledge in a sensible and compact way; clearly, both problems hugely depend on the learning task.

To this regard, in the past, researchers have focused their efforts on creating new algorithms to get models from an a-priori fixed representation. Recently, the attention has drifted towards approaches which try to learn also the optimal representation.

Nowadays, the most clear example of this new trend are the *deep neural networks* (DNNs). The success of the deep paradigm is mainly due to its ability to create many different levels of abstractions of the input representation, and also in its capability of dealing with highly non-linear functions. However, these capabilities come with a cost: besides their need of high computational power, the main concern is that, even for experts in the field, it is not clear, from a theoretical point of view, why DNNs work. This strong black-box nature makes difficult their application in contexts where the understanding of the model is as important as the quality of the model itself.

An alternative to the (deep) neural network based approaches is represented by the more solid theoretical framework concerning *kernels*.

4.2 Kernel function

Despite kernel functions have been studied since the first years of the twentieth century [Mer09], they became very popular in the ML community thanks to the introduction of the Support Vector Machines (SVM) [BGV92, CV95].

Loosely speaking, a kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a similarity measure between elements in $\mathcal{X}$. As stated in the previous section, the concept of similarity is very important in ML because it allows to generalize to unseen patterns.

A more formal definition of a kernel function is the following.

Definition 4.1 : Kernel function

Given a nonempty set $\mathcal{X}$ and a function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, we say that κ is a kernel on $\mathcal{X} \times \mathcal{X}$ if κ is:

- symmetric, i.e., if $\forall\ \mathbf{x}, \mathbf{z} \in \mathcal{X}, \kappa(\mathbf{x}, \mathbf{z}) = \kappa(\mathbf{z}, \mathbf{x})$, and
- positive-semidefinite, i.e., if $\forall\ \mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}, m \geq 1$ the matrix $\mathbf{K}$ defined as $\mathbf{K}_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ is positive-semidefinite.

The symmetric property is self explanatory, while for the positive-semidefiniteness we need to define when a matrix is defined as such.

Definition 4.2 : Positive Semidefinite Matrix

A matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$ is called positive semi-definite if for all $a_1, \ldots, a_m \in \mathbb{R}$ it satisfies

$$\sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j \mathbf{K}_{i,j} \geq 0, \tag{4.2}$$

or equivalently, if all its eigenvalues are non-negative.

The matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$, defined as $\mathbf{K}_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ for any pair of patterns taken from the set $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, is called *Gram matrix* (or *kernel matrix*) of κ with respect to $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

The properties in the Definition 4.2 are necessary to guarantee that the function κ corresponds to a dot-product in some space $\mathcal{H}$ via a mapping function ϕ,

$$\phi : \mathcal{X} \to \mathcal{H} \tag{4.3}$$

$$\mathbf{x} \mapsto \phi(\mathbf{x}), \tag{4.4}$$

and hence the kernel κ between $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ can be written as:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \phi(\mathbf{x})^\intercal \phi(\mathbf{z}). \tag{4.5}$$

Equation (4.5) shows that κ can be defined as a dot-product and this underline the fact that it represents a similarity, specifically, a similarity measure between vectors in the space $\mathcal{H}$, called *feature space*. An important observation is that, by defining a kernel function, we are implicitly defining a new representation of the inputs. With a slight abuse of notation, in the following we will refer with $\mathcal{H}$ to the space of functionals, called *Reproducing Kernel Hilbert Space* (RKHS). Roughly speaking, given a positive semi-definite kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, then there uniquely exists a RKHS $\mathcal{H}$ consisting of functions on $\mathcal{X}$ such that $\kappa(\cdot, \mathbf{x}) \in \mathcal{H}$ for every $\mathbf{x} \in \mathcal{X}$, and in such space κ owns the so-called reproducing property. For a more in depth explanation about kernel functions and RKHS please refer to [SS01, STC04]. Table 4.1 provides some examples of well known kernel functions. There are also several kernel functions designed for particular applications, such as biology [STV04], natural language processing (NLP) [LSST+02], or for particular type of input data, e.g., graphs [VSKB10].

In many ML approaches normalizing the data can give many benefits since it avoids, for example, scaling problems. For instance, with SVMs a feature with a very large value dominates the other features while computing the kernel. For this reason, it could be useful to work with the normalized version of a kernel matrix. The normalization of a kernel function is calculated by

$$\tilde{\kappa}(\mathbf{x}, \mathbf{z}) = \frac{\kappa(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa(\mathbf{x}, \mathbf{x})\kappa(\mathbf{z}, \mathbf{z})}}, \tag{4.6}$$

or in its matrix form

$$\widetilde{\mathbf{K}} = \frac{\mathbf{K}}{\sqrt{\mathbf{d}\mathbf{d}^\intercal}}, \tag{4.7}$$

where $\mathbf{d}$ is a m-dimensional vector containing the diagonal of $\mathbf{K}$.

4.3 Kernel methods

4.3.1 Basic concepts from statistical learning theory

In Section 4.1 we pointed out that in ML we want to infer the function h which best approximates the underpinning relation f in the empirical data. It is clear that a re-

[5] RBF stands for Radial Basis Function kernel that is also known as Gaussian kernel.

Name	Function $\kappa(\mathbf{x}, \mathbf{z})$	Hyper-parameters
Linear	$\langle \mathbf{x}, \mathbf{z} \rangle$	-
Polynomial	$(\alpha \langle \mathbf{x}, \mathbf{z} \rangle + c)^d$	$\alpha, c \in \mathbb{R}, d \in \mathbb{N}^+$
RBF[5]	$\exp\left(-\gamma \lVert \mathbf{x} - \mathbf{z} \rVert_2^2\right)$	$\gamma \in \mathbb{R}_{\geq 0}$
Sigmoid	$\tanh(\alpha \langle \mathbf{x}, \mathbf{z} \rangle + c)$	$\alpha, c \in \mathbb{R}$

Table 4.1: Examples of kernel functions.

quirement in any learning problem is to specify a criterion according to which we can assess the quality of our approximated function h. This criterion is usually referred to as *loss function* and it can be defined as a function $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ on the triplet $(\mathbf{x}, y, f(\mathbf{x}))$ consisting of a pattern $\mathbf{x}$, a target value y and a prediction $h(\mathbf{x})$ such that $L(\mathbf{x}, y, y) = 0$ for all $\mathbf{x}$ and y.

Since we want to be as much general as possible, we wish to find a prediction function h that minimizes such loss (a.k.a. error) in as much unseen patterns as possible. In other words, we aim to minimize the so-called *true risk* (or test error). Generally, this is hard to address from both practical and computational point of view, and hence it is assumed that the test patterns are unavailable. The best we can do is to estimate such risk by minimizing the *expected risk* over all possible training patterns. However, this is also difficult because it can be done only by knowing the real probability distribution over the patterns, which is in general unknown.

A possible solution is to make an approximation of the expected risk by using only the available training data. This approximation is called *empirical risk*. The difference between the expected risk and the empirical risk is called *estimation error* or *generalization error*.

Definition 4.3 : Empirical risk

The empirical risk of an hypothesis $h : \mathcal{X} \to \mathcal{Y}$ over the training data $\mathcal{D} \equiv \{(\mathbf{x}_i, y_i)\}_{i=1}^l$ is defined as

$$R_{\text{emp}}[h] = \frac{1}{l} \sum_{i=1}^{l} L(\mathbf{x}_i, y, h(\mathbf{x})). \tag{4.8}$$

Now, it is sufficient to find the hypothesis h that minimizes such quantity. This is called *empirical risk minimization* (ERM) *induction principle*.

Unfortunately, if we allow h to be taken from a large class of functions $\mathcal{F}$ we can always find an h with small $R_{\text{emp}}[h]$, but we do not have any guarantee that such h is also a minimizer of the expected risk.

Example of a "learning" fail

Let us consider a binary classification problem with training set $\mathcal{D} \equiv \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, we can choose as our hypothesis the function

$$h(\mathbf{x}) = \begin{cases} y & \text{if } (\mathbf{x}, y) \in \mathcal{D} \\ -1 & \text{otherwise.} \end{cases}$$

Without any learning we defined a function with no empirical risk (by definition), but that actually predicts -1 for all possible new patters.

This observation is at the basis of the popular *No-Free-Lunch Theorem* [Wol96].

A similar issue to the one described in the example above can be obtained by learning a very high degree polynomial function that is able to perfectly fit the data. Even though it has no empirical risk, it is very unlikely that such an unstable function will generalize well to unseen patterns. Both the just mentioned hypothesis are *overfitting* the data: a function h is said to overfit the data if it has very small empirical risk but an high expected risk.

In order to overcome this problem, the main idea, which comes from *statistical learning theory* (or VC-theory) [Vap95], is to limit the "complexity" (capacity) of the set of functions $\mathcal{F}$. VC-theory provides bounds on the test error, as function of both the empirical risk and the capacity of the class function. The minimization of such bounds leads to the *structural risk minimization* principle which is at the basis of the best-known kernel method, namely SVM.

In practice, instead of directly constrain the set $\mathcal{F}$, a *regularization* term is added to the objective function, that is $R_{\text{emp}}[h]$. Such regularization rewards "smoother" functions while penalizes the unstable ones.

Definition 4.4 : Regularized empirical risk

The regularized empirical risk (or regularized risk) of an hypothesis $h : \mathcal{X} \to \mathcal{Y}$ over the training data $\mathcal{D} \equiv \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ is defined as

$$R_{\text{reg}}[h] = R_{\text{emp}}[h] + \lambda \Psi[h], \tag{4.9}$$

where $\lambda \leq 0$ is called regularization parameter, and $\Psi[h]$ represents some compactness measure.

The minimum of $R_{\text{reg}}[h]$ represents an upper bound of the empirical risk. In the following section we will see how this regularized risk are connected to kernel methods.

4.3.2 Kernel machines

It is possible to explicitly characterize the form of a minimizer of $R_{\text{reg}}[h]$ using one of the most known theorem in ML, the *Representer Theorem*, introduced by Kimeldorf et al. [KW71].

Theorem 4.1 : Representer theorem [KW71, HSS08]

Let $\Omega : \mathbb{R}_+ \to \mathbb{R}$ be a strictly monotonic increasing function, and let $\mathcal{X}$ be a nonempty set, and $c : (\mathcal{X} \times \mathbb{R} \times \mathbb{R})^l \to \mathbb{R} \cup \{\infty\}$ be an arbitrary loss function. Then each minimizer $h \in \mathcal{H}$ of the regularized risk functional

$$c((\mathbf{x}_1, y_1, h(\mathbf{x}_1)), \ldots, (\mathbf{x}_l, y_l, h(\mathbf{x}_l))) + \Omega(\|h\|_{\mathcal{H}}), \tag{4.10}$$

admits a representation of the form

$$h(\mathbf{x}) = \sum_{i=1}^{l} \boldsymbol{\alpha}_i \kappa(\mathbf{x}_i, \mathbf{x}), \tag{4.11}$$

where κ is a kernel function and $\boldsymbol{\alpha}_i \in \mathbb{R}, \ \forall\, i \in [l]$.

Theorem 4.3.2 shows that it is possible to express the minimizer of $R_{\text{reg}}[h]$ as a combination of the training examples, via a kernel function.

Broadly speaking, *kernel machines* (a.k.a. kernel methods) are a class of ML methods which minimize some form of R_{reg} and make use of kernels. By exploiting kernels this methods are able to transform non linear problems in the input space into tractable linear ones in some (high dimensional) feature space. Specifically, kernel methods are defined as convex optimization problems on some feature space. If the problem formulation considers only mappings ϕ inside dot-products, the latter can be computed by means of their corresponding kernel functions, and the representer theorem gives a theoretical guarantee that the solution has the form (4.11).

Now, the question is: why are kernels useful? The core idea behind the use of kernel functions is their ability to implicitly compute similarity between examples in a high (potentially infinite) dimensional space. This allows to find linear relations in the feature space which correspond to non linear ones in the input space. This is usually referred

to as *kernel trick*. Let us consider for simplicity a binary classification problem in which patterns are not linearly separable in the input space. Then, thanks to the kernel trick, it is possible to search for a linear hypothesis in the feature space without the burden of expliciting the mapping of the patterns in such space, which could be infeasible.

Figure 4.1 shows an example of binary classification in which the patterns are not linearly separable in the input space (left hand side of the figure). However, thanks to the mapping function ϕ, in the feature space (right hand side of the figure) there exist hyperplanes which are able to separate the data points. In the example, the hyperplane in the feature space corresponds in the input space to an ellipse-shaped boundary that contains only the "white" data points.

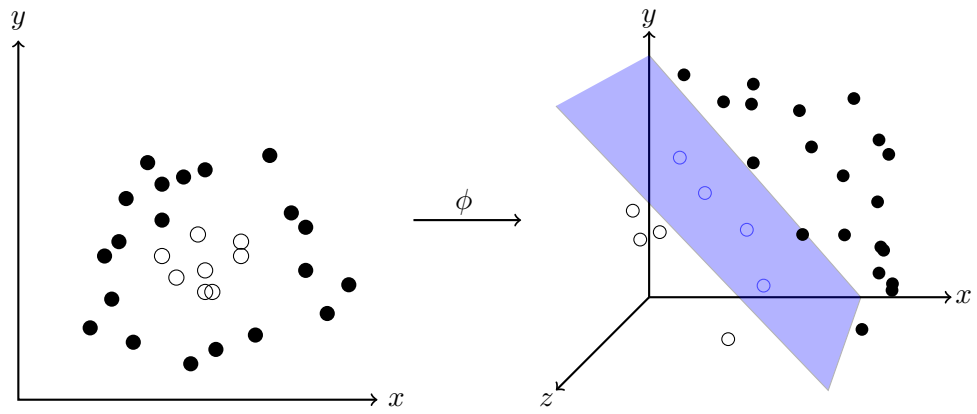


Figure 4.1: Example of binary classification using the kernel trick: (left) in the input space, i.e., $\mathbb{R}^2$, examples are not linearly separable, but after the projection (right), through ϕ, onto an higher dimensional space, i.e., the feature space $\mathbb{R}^3$, examples are separable by, for example, the blue hyperplane.

4.3.3 KOMD

In this section we present a kernel method called KOMD (which stands for *Kernel Optimization of the Margin Distribution*) proposed by Aiolli et al. [ADSMS08]. This classification algorithm has been designed to find the hypothesis which maximizes the margin between positive and negative patterns. The margin of an example is defined as its distance from the decision boundary (i.e., the hypothesis). Assuming a binary classification problem, maximizing the margin means finding the hypothesis which separates positive and negative examples and it is the farthest from the nearest example. Margin theory [Vap95, StC99, GR03] provides good support to the generalization performance of such kind of classifiers (a.k.a. maximum margin classifiers). In Figure 4.2 is depicted an example of maximum margin hypothesis.

The hypothesis is an hyperplane and it can be defined by $\langle \mathbf{w}, \mathbf{x} \rangle + b$, where $\mathbf{w} \in \mathbb{R}^n$

(in the example $n = 2$) and $b \in \mathbb{R}$. The margin is indicated with the greek letter ρ. In red are highlighted the closest points to the decision boundary.

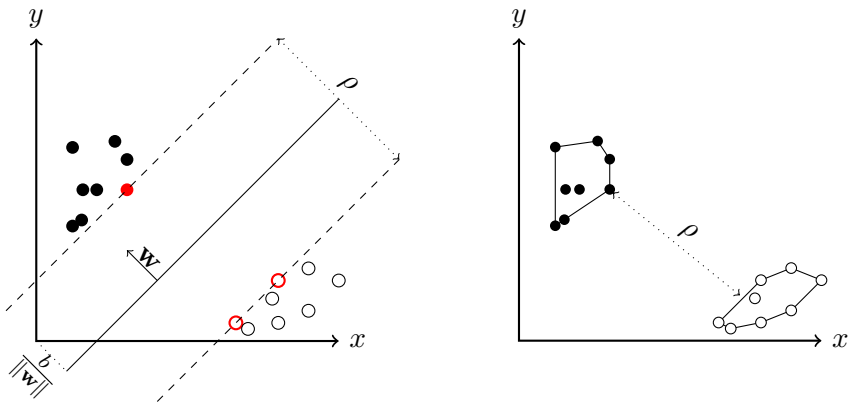


Figure 4.2: Left plot: margin of (for example) an SVM, i.e., ρ, of the hyperplane defined by $\mathbf{w}$, and the offset b. In red are highlighted the examples which lay on the margin. Right plot: margin as defined by KOMD.

In order to describe KOMD, let us consider a binary classification problem with training data $\mathcal{D} \equiv \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, and let $\mathcal{S}_\oplus \equiv \{\mathbf{x} \mid (\mathbf{x}, y) \in \mathcal{D}, y = 1\}$ be the set of positive patterns, and $\mathcal{S}_\ominus \equiv \{\mathbf{x} \mid (\mathbf{x}, y) \in \mathcal{D}, y = -1\}$ be the set of negative patterns. First of all, the method provides a two players zero-sum game interpretation of the hard-margin SVM [BGV92]. Let P_{MIN} and P_{MAX} be the two players of the game. The game takes place in rounds: on each round, P_{MAX} selects an hypothesis h from the hypotheses space $\mathcal{H}$, defined by

$$\mathcal{H} \equiv \{h(\mathbf{x}) : \mathbf{x} \mapsto \langle \mathbf{w}, \phi(\mathbf{x}) \rangle - b \mid \phi : \mathbb{R}^n \to \mathbb{R}^N, \mathbf{w} \in \mathbb{R}^n \text{ s.t. } \|\mathbf{w}\|_2 = 1, b \in \mathbb{R}\}, \quad (4.12)$$

and, simultaneously, P_{MIN} picks a pair of (n-dimensional) patterns $(\mathbf{x}_\oplus, \mathbf{x}_\ominus) \in \mathcal{S}_\oplus \times \mathcal{S}_\ominus$. In the game, the goal of P_{MAX} is to maximize the achieved margin $\rho_h(\mathbf{x}_\oplus, \mathbf{x}_\ominus) = h(\mathbf{x}_\oplus) - h(\mathbf{x}_\ominus)$. On the other hand, P_{MIN} wants to minimize such margin and his strategy consists in choosing the positive and the negative examples according to a probability distribution $\hat{\Gamma}$ over the sets $\mathcal{S}_\oplus$ and $\mathcal{S}_\ominus$, which is taken from the domain of probability distributions

$$\Gamma \equiv \{\boldsymbol{\gamma} \in [0,1]^l \mid \sum_{\mathbf{x}_i \in \mathcal{S}_\oplus} \gamma_i = 1, \sum_{\mathbf{x}_j \in \mathcal{S}_\ominus} \gamma_j = 1\}. \quad (4.13)$$

The value of the game is the expected value of the margin, and this is equivalent to the hard-margin SVM which can be solved by optimizing the following optimization

problem

$$\min_{\boldsymbol{\gamma}\in\hat{\Gamma}} \sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \gamma_i \gamma_j \phi(\mathbf{x}_i)^\intercal \phi(\mathbf{x}_j) = \min_{\boldsymbol{\gamma}\in\hat{\Gamma}} \boldsymbol{\gamma}^\intercal \mathbf{Y}\mathbf{K}\mathbf{Y}\boldsymbol{\gamma}, \tag{4.14}$$

where $\mathbf{K} \in \mathbb{R}^{l\times l}$ is the kernel matrix created over the training patterns such that $\mathbf{K}_{i,j} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$, and $\mathbf{Y} \in \{0, \pm 1\}^{l\times l}$ is a diagonal matrix containing the labels of the examples. Given the optimal solution $\boldsymbol{\gamma}^*$ of (4.14), the value of the (non normalized) weights vector $\mathbf{w}^*$ is given by

$$\mathbf{w}^* = \sum_{(\mathbf{x}_i, y_i)\in\mathcal{D}} y_i \gamma_i^* \phi(\mathbf{x}_i). \tag{4.15}$$

In order to punish unstable solutions (see Section 4.3.1), a regularization term is added to the optimization problem. In particular, the player P_{MIN} is penalized by some extent if the probability distribution $\hat{\Gamma}$, which influences his random choice, has high variance. This regularization is done over the squared L^2-norm of the vector $\boldsymbol{\gamma}$, which leads to the following optimization problem

$$\min_{\boldsymbol{\gamma}\in\hat{\Gamma}} (1-\lambda)\boldsymbol{\gamma}^\intercal \mathbf{Y}\mathbf{K}\mathbf{Y}\boldsymbol{\gamma} + \lambda\|\boldsymbol{\gamma}\|_2^2, \tag{4.16}$$

with $\lambda \in [0, 1]$ the regularization parameter.

> **Observations on λ**
>
> It is easy to see that anytime $\lambda = 0$, the problem (4.16) is indeed the hard-margin SVM (4.14). At the other extreme, that is $\lambda = 1$, the problem becomes $\min_{\boldsymbol{\gamma}\in\hat{\Gamma}} \|\boldsymbol{\gamma}\|_2$, which returns the squared distance between the centroid of the convex hull of the positive and the negative examples in the feature space.

Once the training is over, i.e., the optimal $\boldsymbol{\gamma}^*$ has been found, the evaluation on a new pattern $\mathbf{x}$ is computed by:

$$h^*(\mathbf{x}) = \langle \mathbf{w}^*, \phi(\mathbf{x})\rangle = \sum_{(\mathbf{x}_i, y_i)\in\mathcal{D}} y_i \gamma_i^* \kappa(\mathbf{x}_i, \mathbf{x}), \tag{4.17}$$

as given by the Representer theorem (see Theorem 4.3.2).

Similarly to the hard-margin SVM, the threshold b is defined as the score of the point which is in the middle between the optimal points in the convex hull of the positive and the negative examples in the feature space. Finally, the classification of a new pattern $\mathbf{x}$

is made according to $\text{sign}(h^*(\mathbf{x}) - b)$.

4.4 Boolean kernels

In this section we present a family of kernels, called Boolean kernels, designed to learn Boolean formulas. As far as we know, the first who introduced the idea of Boolean kernel was Ken Sadohara [Sad01]. In this work the concept of Boolean kernel is actually related to a single kernel called DNF kernel. Specifically, Sadohara proposed an SVM for learning Boolean functions: since every Boolean (i.e., logical) functions can be expressed in terms of Disjunctive Normal Form (DNF) formulas, the proposed kernel creates a feature space containing all possible conjunctions of negated or non-negated Boolean variables.

> **Disjunctive Normal Form**
>
> In logic, a Disjunctive Normal Form (DNF) is a normalization of a logical formula which is a disjunction of conjunctive clauses. For example, given the Boolean variables $x_1, \dots x_4 \in \{0, 1\}$, $(x_1 \wedge x_2 \wedge x_3) \vee x_2 \vee (x_1 \wedge x_4)$ is a valid DNF formula.
>
> In particular, the one presented above is a *monotone* DNF since it has only variables in their positive form. A non-monotone DNF, or simply DNF, can contain negated variables, e.g., $(\neg x_1 \wedge x_2 \wedge \neg x_3) \vee x_2 \vee (\neg x_1 \wedge x_4)$.

For instance, the feature space for a two variables, e.g., x_1, x_2, DNF contains the following $3^2 - 1$ features:

$$x_1, x_2, \neg x_1, \neg x_2, x_1 \wedge x_2, x_1 \wedge \neg x_2, \neg x_1 \wedge x_2, \neg x_1 \wedge \neg x_2, \tag{4.18}$$

which can be expressed in mathematical form as

$$x_1, x_2, 1 - x_1, 1 - x_2, x_1 x_2, x_1(1 - x_2), (1 - x_1)x_2, (1 - x_1)(1 - x_2). \tag{4.19}$$

In this way, the resulting decision function of a kernel machine which employs the DNF kernel (as described in Section 4.3) can be represented as a weighted linear sum of conjunctions, which in turn can be seen as a kind of "soft" DNF.

Formally, the DNF kernel between $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$ is defined as

$$\kappa_{\text{DNF}}(\mathbf{x}, \mathbf{z}) = -1 + \prod_{i=1}^{n} (2\mathbf{x}_i\mathbf{z}_i - \mathbf{x}_i - \mathbf{z}_i + 2), \tag{4.20}$$

while its monotone (i.e., without negations) form is the following

$$\kappa_{\text{mDNF}}(\mathbf{x}, \mathbf{z}) = -1 + \prod_{i=1}^{n}(\mathbf{x}_i \mathbf{z}_i + 1). \tag{4.21}$$

By restricting the domain of the vectors in $\{0,1\}^n$ the computation of the kernels is simplified as follows

$$\kappa_{\text{DNF}}(\mathbf{x}, \mathbf{z}) = -1 + 2^{\langle \mathbf{x}, \mathbf{z} \rangle + \langle \overline{\mathbf{x}}, \overline{\mathbf{z}} \rangle}, \tag{4.22}$$

$$\kappa_{\text{mDNF}}(\mathbf{x}, \mathbf{z}) = -1 + 2^{\langle \mathbf{x}, \mathbf{z} \rangle}, \tag{4.23}$$

where $\overline{\mathbf{x}} = \mathbf{1}_n - \mathbf{x}$ is the vector with all the binary entries swapped. Note that the sum $\langle \mathbf{x}, \mathbf{z} \rangle + \langle \overline{\mathbf{x}}, \overline{\mathbf{z}} \rangle$ simply counts the number of common "bits" between $\mathbf{x}$ and $\mathbf{z}$. These last two kernels were independently discovered in [Wat99] and [KRS01].

One of the drawbacks of these kernels is the exponential growth of the feature space with the number of involved variables, i.e., $3^n - 1$ for n variables. In order to have more control on the size of the feature space, Sadohara et al. [Sad02] proposed a reduced variation of the DNF kernel in which only conjunctions with up to d variables (i.e., d-ary conjunctions) are considered. This kernel is called d-DNF kernel, and on binary vectors it is defined as

$$\kappa^d_{\text{DNF}}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{d} \binom{\langle \mathbf{x}, \mathbf{z} \rangle + \langle \overline{\mathbf{x}}, \overline{\mathbf{z}} \rangle}{i}, \tag{4.24}$$

and clearly, if $d = n$, $\kappa^d_{\text{DNF}}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{DNF}}(\mathbf{x}, \mathbf{z})$. A nice property of the d-DNF kernel is that it yields a nested sequence of hypothesis spaces, i.e., $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \cdots \subseteq \mathcal{H}_n$. Thus, choosing a degree d (a.k.a. arity) for the kernel implicitly means controlling the capacity of the hypothesis space, which is a very important aspect in learning (see Section 4.3). The same idea can also be applied to the monotone DNF kernel [NN14]:

$$\kappa^d_{\text{mDNF}}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{d} \binom{\langle \mathbf{x}, \mathbf{z} \rangle}{i}. \tag{4.25}$$

Instead of limiting the number of involved variables, Zhang et al. [ZLKY03] proposed a parametric version of the DNF and mDNF kernel. Specifically, given $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$ and

$\sigma > 0$, then

$$\kappa^{(\sigma)}_{\text{DNF}}(\mathbf{x}, \mathbf{z}) = -1 + \prod_{i=1}^{n}(\sigma \mathbf{x}_i \mathbf{z}_i + \sigma(1 - \mathbf{x}_i) + \sigma(1 - \mathbf{z}_i) + 1), \tag{4.26}$$

$$\kappa^{(\sigma)}_{\text{mDNF}}(\mathbf{x}, \mathbf{z}) = -1 + \prod_{i=1}^{n}(\sigma \mathbf{x}_i \mathbf{z}_i + 1). \tag{4.27}$$

The purpose of the parameter σ is to induce an inductive bias towards simpler or more complex DNF formulas. In particular, values of σ in the range $[0, 1]$ give a bias towards shorter DNF, while for $\sigma > 1$ the bias is more towards complex DNF. When $\sigma = 1$, then $\kappa^{(\sigma)}_{\text{DNF}}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{DNF}}(\mathbf{x}, \mathbf{z})$, and the same for the monotone DNF kernel. In this work the authors also showed that, for binary vectors, the polynomial kernel, i.e., $\kappa^{p}_{\text{POLY}}(\mathbf{x}, \mathbf{z}) = (\sigma\langle \mathbf{x}, \mathbf{z}\rangle + c)^p$, $c \in \mathbb{R}$, is a Boolean kernel, even though they did not provide any formal definition of Boolean kernel. However, an important observation is that the feature space of the polynomial kernel is represented by all the monomials (i.e., conjunctions) up to the degree p. So, in some sense, it is similar to the d-DNF kernel, the core difference is that the polynomial associates different weights to the features. It is worth to notice that the polynomial kernel contains sets of equivalent features in its embedding, e.g., with $p = 3$ and $\mathbf{x} \in \{0, 1\}^2$ we would have the features $\mathbf{x}_1^2\mathbf{x}_2, \mathbf{x}_1\mathbf{x}_2^2$ that are the same feature $\mathbf{x}_1\mathbf{x}_2$.

In a subsequent work [ZLC05], the same authors proposed a decision rule classifier called DRC-BK, which learns a decision hyperplane through an SVM with Boolean kernels, specifically $\kappa^{(\sigma)}_{\text{mDNF}}$, and then it mines classification rules from this hyperplane.

A kernel related to the polynomial is the all-subset kernel [STC04, KT14], defined as

$$\kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{n}(\mathbf{x}_i \mathbf{z}_i + 1), \tag{4.28}$$

which considers a space with a feature for each subset of the input variables, including the empty subset. It is different from the polynomial because it does not limit the number of considered monomials, and it gives the same weight to all features. It is easy to see that the all-subset kernel and the monotone DNF kernel are actually the same kernel up to the constant -1, i.e., $\kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{mDNF}}(\mathbf{x}, \mathbf{z}) + 1$.

Both the polynomial and the all-subsets kernel have limited control of which features they use and how they are weighted. The polynomial kernel uses only monomials of degree up to p with a weighting scheme depending on a parameter (c). The all-subsets, instead, makes use of the monomials corresponding to all possible subsets of the n input variables.

A restricted version of the all-subset kernel is the ANOVA kernel [STC04] in which

the embedding space is formed by monomials with a fixed degree d without repetition. For example, given $\mathbf{x} \in \{0,1\}^3$ the feature space of the all-subset kernel would be made by the features $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_1\mathbf{x}_2, \mathbf{x}_1\mathbf{x}_3, \mathbf{x}_2\mathbf{x}_3, \mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ and $\emptyset$, while for the ANOVA kernel of degree 2 it would be composed by $\mathbf{x}_1\mathbf{x}_2, \mathbf{x}_1\mathbf{x}_3$ and $\mathbf{x}_2\mathbf{x}_3$. Formally, the ANOVA kernel is defined as follows

$$\kappa_{\text{A}}^{d}(\mathbf{x}, \mathbf{z}) = \sum_{1 \leq i_1 < i_2 < \cdots < i_d \leq n} \prod_{j=1}^{d} \mathbf{x}_{i_j} \mathbf{z}_{i_j}, \tag{4.29}$$

where $i_1, i_2, \ldots, i_d$ are all the possible set of indices of cardinality d, taken from $[n]$.

Another well known (Boolean) kernel is the Tanimoto kernel [RSSB05], computed by

$$\kappa_{\text{T}}(\mathbf{x}, \mathbf{z}) = \frac{\langle \mathbf{x}, \mathbf{z} \rangle}{\|\mathbf{x}\|_2^2 + \|\mathbf{z}\|_2^2 - \langle \mathbf{x}, \mathbf{z} \rangle}, \tag{4.30}$$

which represents the Jaccard similarity coefficient in binary contexts.

From an application point of view, Boolean kernels have been successfully applied on face recognition [CHW08, CD09a], spam filtering [LC09], load forecasting [CD09b], and on generic binary classification tasks [Sad02, ZLKY03].

4.5 Margin, and radius of the MEB

In Section 4.3.3 we introduced the concept of margin and how it can be computed. Let us recall it: given a training Gram matrix $\mathbf{K}$ computed over the training data $\mathcal{D}$, the margin ρ obtained by a margin maximization algorithm, e.g., an hard-margin SVM, can be computed (actually its square) by solving the quadratic programming problem

$$\rho^2 = \min_{\boldsymbol{\gamma} \in \Gamma} \boldsymbol{\gamma}^{\intercal} \mathbf{Y}\mathbf{K}\mathbf{Y}\boldsymbol{\gamma}, \tag{4.31}$$

with Γ and $\mathbf{Y}$ defined as in Section 4.3.3.

Thanks to the theoretical guarantees regarding the generalization power of hypothesis with large margin, the focus of many ML algorithms is concentrated in finding representations able to achieve very large margin. However, there is an aspect that should not be neglected, that is how data points are distributed in the feature space. In order to clarify this concept, let us make an example.

Let us assume of choosing the identity matrix as kernel matrix. By definition this kernel induces an embedding space (i.e., feature space) where each example is mapped

onto an orthogonal dimension w.r.t. to the others. This guarantees that the induced RKHS of functionals contains an hypothesis that separates ($\rho > 0$) the data whatever the labelling is, and at the same time such margin ρ is quite large. Unfortunately, the resulting best hypothesis does not have any generalization ability. This is due to different factors, but one of this is related to the concept of the Minimum Enclosing Ball (MEB), also called minimum enclosing sphere, of the data points.

The MEB, as the name suggests, represents the smallest hypersphere that contains all the data points in the feature space (see Figure 4.3). It is clear that the larger the MEB the higher are the chances of having a large margin too. Yet, defining a feature space that is naïvely large can lead to a situation like the one just described.

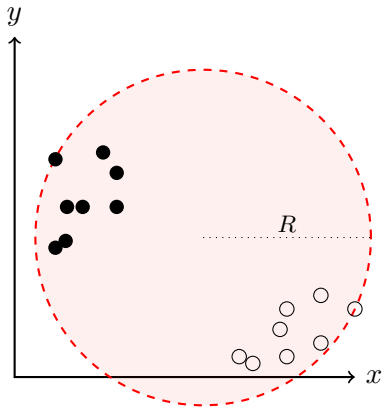


Figure 4.3: Bi-dimensional example of Minimum Enclosing Ball (MEB): R is the radius of the MEB.

For this reason, there exist methods that take into account the size of the MEB besides the margin. In order to calculate the size of a MEB, we need to know its radius (R), which can be computed as in the following

$$R^2 = \max_{\boldsymbol{\alpha} \in \mathcal{A}} \boldsymbol{\alpha}^\intercal \mathbf{d} - \boldsymbol{\alpha}^\intercal \mathbf{K} \boldsymbol{\alpha}, \tag{4.32}$$

where $\mathcal{A} \equiv \{\boldsymbol{\alpha} \in \mathbb{R}^m \mid \sum_{i=1}^m \boldsymbol{\alpha}_i = 1\}$, and $\mathbf{d}$ is m-dimensional vector containing the diagonal of $\mathbf{K}$. With a normalized kernel matrix ($\widetilde{\mathbf{K}}$) the computation of the radius can be simplified as

$$R^2 = 1 - \min_{\boldsymbol{\alpha} \in \mathcal{A}} \boldsymbol{\alpha}^\intercal \widetilde{\mathbf{K}} \boldsymbol{\alpha}, \tag{4.33}$$

since, by definition, $\forall \boldsymbol{\alpha}, \boldsymbol{\alpha}^\intercal \mathbf{d} = \boldsymbol{\alpha}^\intercal \mathbf{1}_m = \|\boldsymbol{\alpha}\|_1 = 1$.

The argumentations about the connection between the margin and the radius we will be resumed and discussed more in depth in Section 4.7.1.

4.6 Expressiveness of kernels

As mentioned in Section 4.2, a kernel function implicitly define a RKHS $\mathcal{H}$ of functionals. Since kernel methods are designed to find a linear hypothesis in $\mathcal{H}$, it is useful to have a way for measuring the discriminatory power of such family of functions. This characteristic is usually called *expressiveness* (expressivity or complexity) of a kernel. Informally, the expressiveness can be defined as the number of different labelings (i.e., dichotomies) that the family of functions, induced by the kernel, is able to shatter. In literature different complexity measures have been proposed such as, the VC-dimension [Vap98, OAR16] and the Rademacher Complexity [KP02, BM03, OGRA15], to mention the most famous ones. Even though they are theoretically well founded and they guarantee strict bounds on the generalization error, these complexity measures are in general computational expensive. For this reason, recently, a very efficient empirical complexity measure have been proposed, namely the *spectral ratio* (SR) [ADNS15]. This measure, which is defined on kernel matrices, represents an empirical approximation of the actual complexity of a kernel.

Formally, given a Gram matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$, the spectral ratio of $\mathbf{K}$, $\mathcal{C}(\mathbf{K})$, is defined as the ratio between the trace (nuclear) norm of the kernel $\mathbf{K}$ and its Frobenius norm, that is

$$\mathcal{C}(\mathbf{K}) = \frac{\|\mathbf{K}\|_{\mathrm{T}}}{\|\mathbf{K}\|_{\mathrm{F}}} = \frac{\sum_{i=1}^{m} \mathbf{K}_{i,i}}{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m} \mathbf{K}_{i,j}^2}}. \tag{4.34}$$

In [Don16] it has been shown the connection between the spectral ratio and the empirical Rademacher complexity. In particular, under mild conditions, given a kernel matrix $\mathbf{K}$ and the class of linear functions $\mathcal{F}$, the following bound holds $\widehat{\mathcal{R}}[\mathcal{F}] \leq \mathcal{O}(\sqrt{\mathcal{C}(\mathbf{K})})$, where $\widehat{\mathcal{R}}[\mathcal{F}]$ stands for the empirical Rademacher complexity of $\mathcal{F}$. Moreover, the (squared) SR is also a lower bound of the rank of the kernel matrix $\mathbf{K}$:

$$1 \leq \mathcal{C}(\mathbf{K}) \leq \sqrt{\mathrm{rank}(\mathbf{K})}. \tag{4.35}$$

Properties of the spectral complexity

The spectral ratio of a kernel matrix owns the following nice properties:

- it is invariant to positive scalar multiplications, i.e., $\forall \alpha \geq 0,\ \mathcal{C}(\alpha \mathbf{K}) = \mathcal{C}(\mathbf{K})$;

- the identity kernel matrix $\mathbf{I}_m$, which has rank m, has the maximal spectral ratio, i.e., $\mathcal{C}(\mathbf{I}_m) = \sqrt{m}$;
- a kernel matrix with all equal (non negative) entries $\overline{\mathbf{K}} = \alpha \mathbf{1}_{m \times m}, \alpha > 0$, which has rank 1, has the minimum spectral ratio, i.e., $\mathcal{C}(\overline{\mathbf{K}}) = 1$.

SR is also connected to the radius of the MEB. In the previous section we showed that, assuming a normalized kernel matrix $\widetilde{\mathbf{K}}$, the radius of the MEB can be computed by (4.33). A good approximation of the radius can be computed as $\widetilde{R}^2(\widetilde{\mathbf{K}}) = 1 - \overline{K}$, where $\overline{K}$ is the average of the entries in the matrix $\widetilde{\mathbf{K}}$. This simple formulation is actually exact in the two extreme cases, that is, $R^2(\mathbf{1}_{m \times m}) = \widetilde{R}^2(\mathbf{1}_{m \times m}) = 0$ and $R^2(\mathbf{I}_m) = \widetilde{R}^2(\mathbf{I}_m) = 1 - 1/m$. Such approximation is a lower bound of the radius. This result shows that the SR can be used as a measure of the intrinsic complexity of the embedding space.

In the following, for measuring the complexity of kernel matrices we will use the standardized version of the spectral ratio, that is defined by

$$\widetilde{\mathcal{C}}(\mathbf{K}) = \frac{\mathcal{C}(\mathbf{K}) - 1}{\sqrt{m} - 1} \in [0, 1]. \tag{4.36}$$

With this normalized version, the identity matrix has complexity 1, while the kernel with all equal entries has complexity 0.

By following the definition as in [ADNS15], we say that a kernel function κ_i is more general than a kernel function κ_j (i.e., $\kappa_i \geq_G \kappa_j$) when $\mathcal{C}(\mathbf{K}_{\mathcal{D}}^{(i)}) \leq \mathcal{C}(\mathbf{K}_{\mathcal{D}}^{(j)})$ for any possible dataset $\mathcal{D}$.

4.7 Multiple Kernel Learning

In their early years, kernel methods made use of kernels that were specifically designed (or chose) to face a particular problem. Usually, once a set of kernels were defined, the best performing one was chosen through a validation step. Even though this is still a valid and effective approach, recently algorithms that automatically learn the best representation, i.e., kernel function, have been proposed.

This new way of facing the representation problem is called Kernel Learning (KL) [CMR10, CKM13, TN04]. One of the most successful KL paradigm is Multiple Kernel Learning (MKL) [BLJ04, SRSS06], in which a combination of different kernels is used instead of a single kernel function [GA11]:

$$\kappa_{\boldsymbol{\mu}}(\mathbf{x}, \mathbf{z}) = f_{\boldsymbol{\mu}}(\{\kappa_i(\mathbf{x}, \mathbf{z})\}_{i=1}^{P}), \tag{4.37}$$

where $f_{\boldsymbol{\mu}} : \mathbb{R}^P \to \mathbb{R}$ is a combination function, and κ_i are the base (a.k.a. weak) kernel functions. From the feature space point of view, the sum of two kernels can be interpret as the concatenation of the features contained in both the RKHS [StC99], and thus the weighted sum of a set of weak kernels can be seen as a weighted concatenation of all the features inside their RKHS.

So, given the combined kernel $\kappa_{\boldsymbol{\mu}}$, its feature space $\mathcal{H}_{\boldsymbol{\mu}}$ is defined by the mapping

$$\mathbf{x} \mapsto \phi_{\boldsymbol{\mu}}(\mathbf{x}) = (\sqrt{\boldsymbol{\mu}_1}\phi_1(\mathbf{x}), \sqrt{\boldsymbol{\mu}_2}\phi_2(\mathbf{x}), \ldots, \sqrt{\boldsymbol{\mu}_P}\phi_P(\mathbf{x})) \in \mathcal{H}_{\boldsymbol{\mu}}, \tag{4.38}$$

where $\phi_i(\mathbf{x})$ is the mapping to the feature space $\mathcal{H}_i$ associated with the kernel κ_i.

MKL algorithms can be mainly used in two ways:

- the set of available kernels correspond to different notions of similarity, and through a learning method the best kernel or the best combination of kernels is automatically selected;
- the kernels (which can be potentially the same) are based on input representations or input data coming from different sources.

Moreover, different learning approaches can be adopted to determine the kernels' combination:

- fixed rule-based [PWCG01, BHN05], in which actually there is not a real learning since a fixed function is used, e.g., sum of kernels or multiplication of kernels;
- heuristic-based [QL09], where a parametrized combination function is used and the parameters are learned through some heuristic measure typically applied to each kernel individually;
- optimization-based [LYL14, AD15], that are the most popular ones, which are those MKL algorithms that learn the parameters by optimizing a specific target function.

In this thesis we focus on non-negative combinations of P weak kernels of the form

$$\kappa(\mathbf{x}, \mathbf{z}) = \sum_{p=1}^{P} \boldsymbol{\mu}_p \langle \phi_p(\mathbf{x}), \phi_p(\mathbf{z}) \rangle = \sum_{p=1}^{P} \boldsymbol{\mu}_p \kappa_p(\mathbf{x}, \mathbf{z}), \ \boldsymbol{\mu}_p \geq 0 \ \forall p \in [P], \tag{4.39}$$

where $\kappa_1, \kappa_2, \ldots, \kappa_P$ are the weak kernels such that $\forall\ p \in [P], \kappa_p(\mathbf{x}, \mathbf{z}) = \langle \phi_p(\mathbf{x}), \phi_p(\mathbf{z}) \rangle$.

4.7.1 Radius-margin ratio optimization

The large part of heuristic-based MKL techniques rely on alignment/similarity between the combined and an ideal kernel, usually defined as $\kappa(\mathbf{x}, \mathbf{z}) = 1$ iff $\mathbf{x}$ and $\mathbf{z}$ come from

the same class, and 0 otherwise. Optimization-based methods, instead, commonly follow the Structural Risk Minimization (SRM) principle and they try to directly maximize the margin in a similar way as in the case of a single kernel. However, it has been shown [GCZ10] that the margin is not the unique important aspect of a good representation, and hence maximizing only the margin can cause scaling problems, initialization problems, or issues related to the convergency of the optimization problem.

The scaling and the initialization problems are related, since they both depend on how kernels are scaled. The scaling issue comes from the parameter optimization: by multiplying one of the weak kernel by a constant $a > 0$, an arbitrary margin can be achieved. A similar issue arises when the kernels are selected, since the same constant multiplication "trick" can be applied prior to the learning. Thus, we can conclude that the margin itself is not enough to measure the quality of kernels [CVBM02, GCZ10].

Once again, Statistical learning theory (SLT) [Vap95, Vap98, VC00] turned out to be a valuable element to overcome this problem. Given a training set with l training examples, SLT provides the following bound on the estimation error ($\mathrm{R}_{\mathrm{est}}$) for single kernel ($\mathbf{K}$) SVM predictor M [GCZ10]:

$$\mathrm{R}_{\mathrm{est}}[M] \leq \frac{1}{l}\sqrt{\mathcal{O}\left(\frac{R^2}{\rho^2}\right)}, \tag{4.40}$$

where R is the radius of the MEB for $\mathbf{K}$ and ρ is the achieved margin.

In support of this result, there is also another bound (similar to (4.40)) which relates the quality of a representation to the radius-margin ratio: given a separable training set $\mathcal{D} \equiv \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, and an hyperplane $(\mathbf{w}, b)$ in the feature space computed by the maximum-margin algorithm M, then for all probability measures $\mathbb{P}$ underlying $\mathcal{D}$ the expectation of the misclassification error (p_{err}) [GM09, CVBM02]

$$p_{\mathrm{err}}(\mathbf{w}, b) = \mathbb{P}(\mathrm{sign}(\mathbf{w}^{\intercal}\phi(\mathbf{X}) + b) \neq \mathbf{y})$$

is bounded by

$$\mathrm{R}_{\mathrm{loo}}[M] = \mathbb{E}\left[p_{\mathrm{err}}(M_{l-1}(\mathcal{D}))\right] \leq \frac{1}{l}\mathbb{E}\left[\frac{R^2(\mathcal{D})}{\rho^2(M(\mathcal{D}), M)}\right],$$

which is the expected value of the Leave-one-out error. Note that p_{err} is taken over the random draw of a training set of size $l-1$ while radius and margin are computed on the random training set of size l.

These results definitely state that the generalization error depends on the ratio between the radius of the MEB and the margin. For this reason, approaches based on the optimization of both the radius of the MEB and the margin have been recently proposed.

Do et al. [DKWH09] proposed R-MKL, a MKL algorithm which encodes the radius

directly into the objective function of a soft-margin L2-SVM. However, the direct optimization of the proposed formulation is unfeasible because it is not convex and thus they provided an approximated convex version. Such algorithm is based on a two phases procedure: in the first phase a quadratic optimization problem over the hyperplane parameters is solved, while the weights of the combined kernels are fixed; in the second phase the weights are optimized through gradient descent. This procedure is repeated until the stopping criteria are met.

In [DK13] a convex formulation of radius-margin based SVM is proposed, however, some approximations have been made in order to make the radius-margin ratio optimization tractable. In [LYL14], instead, the radius information is directly incorporated inside the optimization criterion via the data scattering matrix, which the authors shows has a close relation with the radius of the MEB.

5

Collaborative filtering

> Search is what you do when you're looking for something. Discovery is when something wonderful that you didn't know existed, or didn't know how to ask for, finds you.
>
> *CNN Money, 2006*
> Jeffrey M. O'Brien

In short

This chapter introduces the collaborative filtering (CF) approach for making recommendation. Recommender Systems will be the main real-world application of one of the framework described in this thesis. In this chapter is presented an overview of the work regarding a particular problem in recommender system, namely the one-class collaborative filtering, in which only positive data are known. Finally, we will describe in detail a CF method which will be extended in the following.

5.1 Collaborative filtering and top-N recommendation

Collaborative Filtering (CF) is the *de facto* approach for making personalized recommendation. CF techniques exploit historical information about the user-item interactions in order to improve future recommendations to users. The general idea behind (user-based) CF approaches is to suggest to a user u items that have been appreciated by users that are similar to u, for some notion of similarity. Figure 5.1 shows a simple example of collaborative recommendation.

In the example the system has to make a recommendation about the movie *Kung Fury* to Eve. The two most similar users, i.e., with similar tastes on already seen movies, are Bob and Carol. Since both of them rated *Kung Fury* with a thumb down, the system will not recommend the movie to Eve.

	Inception	**Gladiator**	**Kung Fury**	**Matrix**
Alice	👍	👎	👍	👍
Bob		👍	👎	👎
Carol	👍	👍	👎	
Dave	👎		👍	
Eve	👍	👍	(👎)	👎

Table 5.1: Example of user-based CF recommendation: the rating inside the parenthesis is the recommended one on the basis of the most similar users.

As we have mentioned in Section 2.2, user-item interactions can be explicit or implicit. Explicit feedbacks are unambiguous piece of information about the extent of the user's appreciation to an item, and it is often represented by a rating, such as, a 1 to 5 stars scale or a thumbs up vs. thumbs down. Contrarily, an implicit feedback is a binary information since it means the presence or the absence of a user-item interaction, and it is by nature ambiguous.

The explicit feedback setting have got most of the researchers attention, however, recently the focus is constantly drifting towards the implicit one (a.k.a. One-Class CF problem, OC-CF). This is due to two main reasons: (i) implicit data are much easier to collect as they do not require any active action by the user: the system is continuously monitoring the user's actions; (ii) they are simply more common: the user is not always willing to give to the system an explicit opinion.

In the implicit feedback setting the goal of a recommender is to produce an ordered list of items, where those items that are the most likely to have a future interaction with the user are positioned at the top (top-N recommendation). Top-N recommendation finds application in many different domains such as TV, movies [HKV08], books [Aio14], music [Aio13, TBCH11], social media [WWB+13] and so on.

The first approaches for OC-CF problems were non-learning neighbourhood-based methods [DK11]. Despite this kind of methods do not employ any learning, they have been shown to be effective on different recommendation scenarios [Aio13, PA16]. More recently, many learning methods for OC-CF have been proposed. These algorithms are part of the bigger family of algorithms called model-based CF methods. In the last two decades, a particular attention has been devoted to latent factor models which try to factorize the rating matrix into two low-rank matrices, $\mathbf{R} = \mathbf{W}\mathbf{X}$, where $\mathbf{W} \in \mathbb{R}^{n \times k}$ represent the user-factors and $\mathbf{X} \in \mathbb{R}^{k \times m}$ the item-factors. These (k) factors can be seen as "meta-features" that define the user tastes ($\mathbf{w}_u \in \mathbf{W}$), and how much of these features are present in the item ($\mathbf{x}_i \in \mathbf{X}$). Usually these methods are referred to as matrix factorization techniques. The prediction of the score of an item for a specific

user is simply calculated by the dot-product of the corresponding row and column in the factorized matrices.

One of the most successful factorization approach for implicit feedback, dubbed WRMF (Weighted Regularized Matrix Factorization), is presented in [HKV08]. In this work Hu et al. proposed an adaptation of the classic SVD (Singular Value Decomposition) method in which it minimizes the square-loss using two regularization terms in order to avoid overfitting. The optimization criterion is defined as:

$$\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{C}_{u,i} (\mathbf{w}_u^\intercal \mathbf{x}_i - 1)^2 + \lambda \|\mathbf{W}\|^2 + \lambda \|\mathbf{X}\|^2, \tag{5.1}$$

where $\mathbf{C}_{u,i}$ are a-priori weights for each pair (u, i) such that positive feedbacks have higher weights than negative ones. Despite being designed for top-N recommendation tasks, this method uses information about the rating values (not binary) to give more importance to user-item interactions with high rating values. In fact, $\mathbf{C}_{u,i}$ is calculated by: $\mathbf{C}_{u,i} = 1 + \alpha \mathbf{R}_{u,i}$, where α is a parameter of the method. In their experiments the best performances have been achieved with $\alpha = 40$.

In [RFGST09], Rendle et al. proposed a Bayesian Personalized Ranking (BPR) criterion, that is the maximum posterior estimator derived from a Bayesian analysis. In particular, BPR addresses the OC-CF problem by turning it into a ranking problem and it assumes that users prefer items they have already interacted with in the past. The overall goal of this method is to find a personalized total ranking ($>_u$) for any user and pair of items. To determine the personalized ranking for any $i \in \mathcal{I}$, BPR aims to maximize the posterior probabilities

$$\mathbb{P}(\Theta | >_u) \propto \mathbb{P}(>_u | \Theta) \mathbb{P}(\Theta), \tag{5.2}$$

where Θ are the parameters of the model. The optimization of Θ is performed through a criterion, called BPR-OPT, which has connection to the AUC metric and optimizes it implicitly. Authors finally show how to adopt this criterion for kNN (BPRkNN) and matrix factorization methods (BPRMF).

In [NK11], Ning et al. presented the method SLIM (Sparse LInear Method) which learns a sparse coefficient matrix for the items in the system solely from the user rating profiles, by solving a regularized optimization problem. Specifically, the optimization problem is defined as:

$$\begin{aligned} &\min_{\mathbf{W}} \frac{1}{2} \|\mathbf{R} - \mathbf{R}\mathbf{W}\|_{\mathrm{F}}^2 + \beta \|\mathbf{W}\|_{\mathrm{F}}^2 + \lambda \|\mathbf{W}\|_1 \\ &\text{s.t. } \mathbf{W} \geq 0 \\ &\quad \mathrm{diag}(\mathbf{W}) = 0. \end{aligned}$$

Since the column of $\mathbf{W}$ are independent to each other this optimization problem can be divided into n sub-optimization problems, one for each column of $\mathbf{W}$ as described in [NK11]. Recently, an extension of the SLIM method, dubbed GLSLIM [CK16], has been proposed.

Even though the methods mentioned above can seem very different each others, they all exploit linear relations between users and/or items. The effectiveness of linear models in collaborative filtering scenarios is further underlined in [BCS14] where Bresler et al. proposed an online linear model and they demonstrated its performance guarantees.

Moreover, in 2013, the winner of the remarkable challenge organized by Kaggle, the Million Songs Dataset challenge [MBMEL12], was an extension of the well known (linear) item-based nearest-neighbors (NN) algorithm [DK04]. This extension [Aio13] (that we call here MSDw) introduced an asymmetric similarity measure, dubbed asymmetric cosine. In a classic item-based CF method, the scoring function for a user-item pair (u, i) is computed by a weighted sum over the items liked by u in the past, that is:

$$\hat{\mathbf{r}}_{u,i} = \sum_{j \in \mathcal{I}} \mathbf{W}_{i,j} \mathbf{R}_{u,j} = \sum_{j \in \mathcal{I}_u} \mathbf{W}_{i,j},$$

where $\mathbf{W}_{i,j}$ expresses the similarity between item i and item j.

As said previously, one of the main contribution of [Aio13], is the asymmetric cosine (asymC) similarity. The intuition behind asymC comes from the connection between the cosine similarity and the conditional probability. In particular, the cosine similarity over a pair of objects (a, b) can be expressed as the square root of the product of the reciprocal conditional probabilities. Let $\mathbf{a}, \mathbf{b} \in \{0, 1\}^n$ be the binary vector representations of items a and b, respectively, then the cosine similarity is:

$$S_{\frac{1}{2}}(a, b) = \cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\intercal \mathbf{b}}{\|\mathbf{a}\|_2 \cdot \|\mathbf{b}\|_2} = \mathbb{P}(a|b)^{\frac{1}{2}} \mathbb{P}(b|a)^{\frac{1}{2}}.$$

The idea of the asymmetric cosine similarity is to give different weights to the conditional probabilities, that is

$$S_\alpha(a, b) = \frac{\mathbf{a}^\intercal \mathbf{b}}{\|\mathbf{a}\|_2^{2\alpha} \|\mathbf{b}\|_2^{1-\alpha}} = \mathbb{P}(a|b)^\alpha \mathbb{P}(b|a)^{1-\alpha},$$

with $\alpha \in [0, 1]$. In case of binary ratings this asymmetric similarity can be computed as in the following. Let $\mathcal{U}_i$ be the set of users who rated the item i, then the asymC between item i and item j is defined by:

$$\mathbf{W}_{i,j} = S_\alpha(i, j) = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{|\mathcal{U}_i|^\alpha |\mathcal{U}_j|^{1-\alpha}}.$$

Besides its outstanding performance in terms of mAP@500, the MSD winning solution is also easily scalable to very large datasets.

More direct approaches for building good rankings over the items are the so-called *learning to rank* [ZTTY+07] methods. Learning to rank approaches exploit supervised machine learning to solve ranking problems. These methods can be divided into three macro categories: *pointwise* approaches [WWB+13, BSR+05, ODNDSM13] in which for each user-item pair a score is predicted and then it is used to build the ranking (very similar to the classic CF approaches); *pairwise* approaches [ZPX+14, RF14] face the ranking problem as a binary classification problem in which they try to minimize the number of inversions in the ranking; *listwise* methods [SLH10, HWL+15] try to directly optimize one of the ranking evaluation measures. The big challenge here is the fact that most of the measures are not continuous functions with respect to the parameters of the model and hence some approximations have to be used.

5.2 CF-OMD

In this section we present the seminal CF algorithm for top-N recommendation, called CF-OMD (Optimization of the Margin Distribution) [Aio14], upon which our kernel-based framework (presented in Chapter 10) is based. CF-OMD is inspired by preference learning, and designed to explicitly maximize the AUC (see Chapter 3) [Aio05, ADSMS08]. This method is related to KOMD presented in Section 4.3.3.

In the following we assume of having a dataset of n users and m items. Let the matrix $\mathbf{W} \in \mathbb{R}^{k \times n}$ be the embeddings of users (arranged in the columns) in a latent factor space, and $\mathbf{X} \in \mathbb{R}^{k \times m}$ be the embeddings of items (arranged in the columns) in such space. Given a user u, a ranking over the items can be induced by the factorization $\widehat{\mathbf{R}} = \mathbf{W}^\intercal \mathbf{X}$, where $\hat{\mathbf{r}}_{u,i} = \mathbf{w}_u^\intercal \mathbf{x}_i$, where $\mathbf{w}_u$ is a column of $\mathbf{W}$ corresponding to the user u and $\mathbf{x}_i$ is a column of $\mathbf{X}$ corresponding to the item i.

Let us now to fix the item representation as $\mathbf{x}_i = \mathbf{r}_i / \|\mathbf{r}_i\|_2$, and let $\rho(i \prec_u j) = (\hat{\mathbf{r}}_{u,i} - \hat{\mathbf{r}}_{u,j})/2 = \mathbf{w}_u^\intercal (\mathbf{x}_i - \mathbf{x}_j)/2$ be the margin for an item pair (i, j) for user u, with $\|\mathbf{w}\|_2 = 1$. Finally, let us also define the probability distribution over the positive and negative items for u as

$$\mathbf{A}_u = \{\boldsymbol{\alpha} \in \mathbb{R}_+^m | \sum_{i \in \mathcal{I}_u} \boldsymbol{\alpha}_i = 1, \sum_{i \notin \mathcal{I}_u} \boldsymbol{\alpha}_i = 1\}. \tag{5.3}$$

In [Aio14] it is proposed an approach to maximize the minimum margin inspired by preference learning where the ranking task is posed as a two-players zero-sum game similar to the one presented in Section 4.3.3.

Let P_{MAX} and P_{MIN} be the players of the game: on each round, P_{MIN} picks a preference

$i \prec_u j$ and, simultaneously, P_{MAX} picks an hypothesis $\mathbf{w}_u$ with the aim of maximizing the margin $\rho(i \prec_u j)$. Let us now suppose that the strategy of P_{MIN} is to choose the positive-negative pair of items on the basis of the probability distribution described by $\mathbf{A}_u$. Then, assuming the picking events independent to each others, the probability of a pair (i, j) is given by $\boldsymbol{\alpha}_i \boldsymbol{\alpha}_j$, for $i \in \mathcal{I}_i, j \notin \mathcal{I}_j$. Thus, the value of the game, i.e., the expected margin, can be computed by:

$$\mathbb{E}_{\boldsymbol{\alpha}}[\rho] = \frac{1}{2} \sum_{i \in \mathcal{I}_i, j \notin \mathcal{I}_j} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j \left(\mathbf{w}_u^\intercal \mathbf{x}_i - \mathbf{w}_u^\intercal \mathbf{x}_j \right), \tag{5.4}$$

and consequently, when P_{MIN} is free to choose any possible strategy, the equilibrium of the game is given by

$$(\mathbf{w}_u^*, \boldsymbol{\alpha}_u^*) = \arg \max_{\|\mathbf{w}\|_2 = 1} \min_{\boldsymbol{\alpha} \in \mathbf{A}_u} \mathbb{E}_{\boldsymbol{\alpha}} [\rho]. \tag{5.5}$$

We can now reformulate, through simple mathematical derivations, the expected margin as:

$$\mathbb{E}_{\boldsymbol{\alpha}}[\rho] = \frac{1}{2} \mathbf{w}_u^\intercal \mathbf{X} \mathbf{Y} \boldsymbol{\alpha}, \tag{5.6}$$

where $\mathbf{Y}$ is a diagonal matrix, $\mathbf{Y} = \text{diag}(\mathbf{y})$, such that $\mathbf{y}_i = 1$ if $i \in \mathcal{I}_u$, -1 otherwise. It can be demonstrated that the $\mathbf{w}_u^*$ which maximizes the expected margin is equal to $\mathbf{w}_u^* = \mathbf{X} \mathbf{Y} \boldsymbol{\alpha}$ normalized. As observed for KOMD in Section 4.3.3, the pure maximization of the minimum margin can lead to poor solutions in terms of generalization capability. Especially in the implicit feedback context, noise is always present in the negative set of items. So, two quadratic regularization terms that impose a bias towards uniform solutions for $\boldsymbol{\alpha}$ are introduced.

Finally, to each regularization term we associate a regularization parameter (λ_p, λ_n) and the following optimization problem is defined in order to maximize the expected margin:

$$\boldsymbol{\alpha}_u^* = \underset{\boldsymbol{\alpha} \in \mathbf{A}_u}{\text{argmin}} \ \boldsymbol{\alpha}^\intercal \left(\mathbf{Y} \mathbf{X}^\intercal \mathbf{X} \mathbf{Y} + \boldsymbol{\Lambda} \right) \boldsymbol{\alpha}, \tag{5.7}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix such that $\boldsymbol{\Lambda}_{ii} = \lambda_p$ if $i \in \mathcal{I}_u$, otherwise $\boldsymbol{\Lambda}_{ii} = \lambda_n$. In this formulation, λ_p and λ_n are regularization parameters $(\lambda_p, \lambda_n \geq 0)$ which give more influence to positive examples and negative examples, respectively.

PART II

New Boolean kernel framework

Summary

6

A new Boolean kernel framework

> No great discovery was ever made without a bold guess.
>
> Isaac Newton

In short

This chapter presents one of the main contribution of this thesis. The new family of kernels that we will describe here owns the characteristic of creating feature spaces that are very easy to interpret, since they are based on logic. Specifically, features are logical formulas (of a fixed form) over the input Boolean variables. This makes the solutions of kernel machines, based on such kernels, understandable because they are combinations of logical rules. Moreover, an extensive evaluation on several categorical datasets shows how Boolean kernels are able to achieve state-of-the-art performance on binary classification tasks.

6.1 Preliminaries

First of all, since in literature a formal definition of Boolean kernel is missing, let us formally define this concept.

Definition 6.1 : Boolean kernel function

A Boolean kernel function is a kernel function $\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$ such that:

- $\mathbf{x}, \mathbf{z} \in \{0,1\}^n, n \in \mathbb{N}_{>0}$;
- the embedding function $\phi : \{0,1\}^n \to \{0,1\}^N, N \in \mathbb{N}_{>0}$, maps the input variables in a feature space composed by logical formulas over the input Boolean variables.

It is worth to notice that, if we restrict the input to binary vectors, all the kernels presented in Section 4.4 are compliant with this definition, with the only exceptions of $\kappa^{(\sigma)}_{\text{mDNF}}$ and $\kappa^{(\sigma)}_{\text{DNF}}$ when $\sigma \neq 1$.

Another Boolean kernel, that we did not mention before, is the linear kernel, i.e., $\kappa_{\text{LIN}}(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$, in which the features correspond to the Boolean literals themselves. This kernel simply counts how many active Boolean variables the input vectors have in common.

Definition 6.1 implicitly provides a very simple interpretation of what kind of similarity a Boolean kernel function $\kappa(\mathbf{x}, \mathbf{z})$ computes, that is, it counts how many logical propositions of a fixed form over the input variables are satisfied in both input vectors.

6.2 Boolean kernels for interpretable kernel machines

As mentioned in the introduction, the main goal of this new family of Boolean kernels is to construct feature spaces that are to some extent easy to interpret. Logic is a language that humans are used to handle, and it is often used to explain reasoning and facts. Decision trees, for example, are very appreciated because they naturally provide a straightforward interpretation of their underlying model: every node is a simple logic rule over the input attributes, and a path from the root to a leaf is a conjunction of such rules.

Figure 6.1 shows an example of a very simple decision tree. Even without any explicit description, it is easy to grasp what it represents. The task is the prediction of whether any kind of accident is going to happen. The involved variables are: the taken route (A or B), the speed of the vehicle (less or more than 70 km/h) and if it is raining or not. On the basis of these conditions the value of the leaves indicate the probability of no accidents.

Beyond the fact that the example is trivial and not very interesting, it is clear that interpreting a decision tree is easy, and in many applications it is a desirable (if not necessary) feature for a learning model. Unfortunately, since they are very easy models, from an accuracy point of view, usually, decision trees are not the best choice, and hence

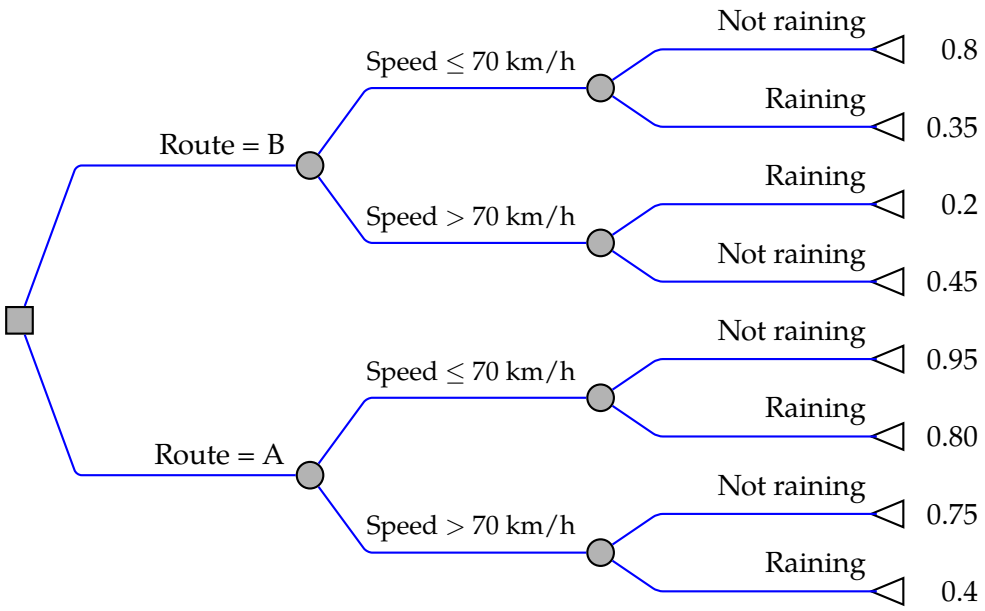


Figure 6.1: A very simple decision tree: internal nodes, i.e., the rules over the variables, are drawn as circles, while the leaves, i.e., the predicted label, are depicted as triangles. The values associated to the leaves indicate the confidence of the decision.

this nice feature of their becomes much less useful.

On the other hand, kernel methods have demonstrated to achieve state-of-the-art performance on many tasks, but they are considered as black-boxes. For example, let us take the output of a SVM: the decision function has this form (as given by the Representer Theorem 4.3.2):

$$f(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{S}} y_i \boldsymbol{\alpha}_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle = \sum_{\mathbf{x}_i \in \mathcal{S}} y_i \boldsymbol{\alpha}_i \kappa(\mathbf{x}, \mathbf{x}_i), \tag{6.1}$$

where $\mathcal{S}$ is the set of so-called *support vectors* (the only ones with a weight $\boldsymbol{\alpha}_i > 0$) [SS01]. The meaning behind this function is not as clear as for the decision tree. Even if we were able in some way to extract the most relevant features, it is unlikely that we could give them a reasonable meaning. What we can observe is that this difficulty of interpretation hugely depends on the kernel function and its induced embedding space. More complex the space more difficult the interpretation.

For this reason, we propose a set of kernels based on propositional logic which induce easy-to-interpret feature spaces. Before digging into the description of the kernels, we need to define some special notations.

Throughout the chapter, whether it is not specified differently, we will refer to vectors $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$, and with a slight abuse of notation, we use the same notation to refer to

the sets interpretation of those vectors: an element (i.e., variable) is contained in the set if it is true in the vector. With $\mathbf{\Phi}$ we indicate the universal set (the set corresponding to the vector $\mathbf{1}_n$) with cardinality n. Given a set $\mathcal{A}$ we refer to its i-th element with $\mathcal{A}_i$ for some enumeration of the elements of $\mathcal{A}$, and with $[\mathcal{A}]^k \equiv \{\mathcal{S} \subseteq \mathcal{A} \mid |\mathcal{S}| = k\}$ we express the set of all the subsets of $\mathcal{A}$ of cardinality k.

Finally, it is noteworthy that for any binary vector $\mathbf{x}$ holds $\langle \mathbf{x}, \mathbf{x} \rangle = \|\mathbf{x}\|_2^2 = \|\mathbf{x}\|_1$ which is the number of ones contained in it, i.e., the cardinality of its set interpretation. For the sake of brevity we will use the notation $\|\mathbf{x}\|$ when $\mathbf{x}$ is considered a vector and $|\mathbf{x}|$ when it is interpreted as a set.

6.3 Monotone Boolean kernels

A Boolean formula (or function) $f : \{0,1\}^n \to \{0,1\}$ is called *monotone* if it does not contain any *not* operator. In other words, f is monotone if by replacing a 0 (i.e., false) with a 1 (i.e., true) its truth value can only change from false to true, i.e., f's value can only increase.

6.3.1 Monotone Literal kernel

In logic, a literal is an atomic formula or its negation. Since we are treating monotone functions, we have only literals in their positive form.

In a Boolean vector, the literals are the variables themselves, and hence the embedding function $\phi_{\text{L}} : \{0,1\}^n \to \{0,1\}^n$ is the identity. Consequently, the monotone Literal (L) kernel, $\kappa_{\text{L}}(\mathbf{x}, \mathbf{z}) = \langle \phi_{\text{L}}(\mathbf{x}), \phi_{\text{L}}(\mathbf{z}) \rangle$, counts how many *true* (i.e., positive) input literals the vectors have in common. Actually, κ_{mL} is exactly the linear kernel $\kappa_{\text{LIN}}(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$, which simply performs the dot-product between the input Boolean vectors.

6.3.2 Monotone Conjunctive kernel

In Boolean algebra, given two variables $x, z \in \{0,1\}$, the conjunction (i.e., *and*) between x and z, denoted by $x \wedge z$, is satisfied if and only if $x = z = 1$, that is if and only if both variables are *true*.

Given two Boolean vectors $\mathbf{x}$ and $\mathbf{z}$, the monotone Conjunctive (mC) kernel $\kappa^c_{\text{mC}}(\mathbf{x}, \mathbf{z})$ counts how many monotone conjunctions, of a fixed arity c, of the variables are *true* (i.e., equals to 1) in both $\mathbf{x}$ and $\mathbf{z}$.

Formally, the embedding of the mC-kernel of degree $c \in [n]$ is given by

$$\phi^c_{\text{mC}} : \mathbf{x} \mapsto (\phi^{(\mathbf{b})}_{\text{mC}}(\mathbf{x}))_{\mathbf{b} \in \mathbb{B}_c}, \tag{6.2}$$

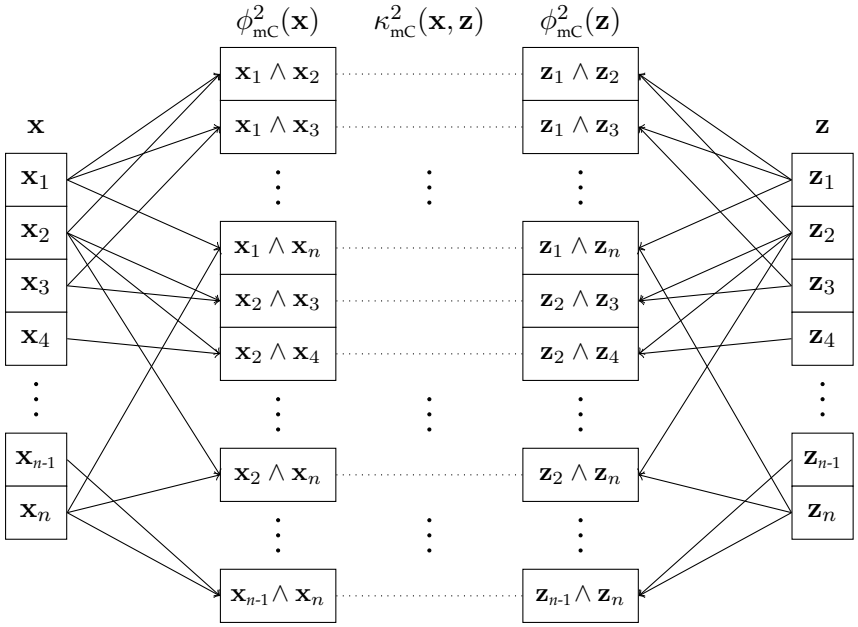


Figure 6.2: Depiction of the mC-kernel of degree 2: firstly the input vectors are mapped into the feature space which is formed by all conjunctions with arity 2 (without repetition). Then, the kernel is computed by "matching" (dotted lines) the corresponding features. Arrows from the input vectors to the feature vectors indicate when a variable influences the conjunction.

where $\mathbb{B}_c = \{\mathbf{b} \in \{0,1\}^n \mid \|\mathbf{b}\|_1 = c\}$, and

$$\phi^{(\mathbf{b})}_{\text{mC}}(\mathbf{x}) = \prod_{i=1}^{n} \mathbf{x}_i^{\mathbf{b}_i} = \mathbf{x}^{\mathbf{b}}, \tag{6.3}$$

where the notation $\mathbf{x}^{\mathbf{b}}$ stands for $\mathbf{x}_1^{\mathbf{b}_1}\mathbf{x}_2^{\mathbf{b}_2}\cdots\mathbf{x}_n^{\mathbf{b}_n}$.

The dimension of the embedding is $\binom{n}{c}$, that is the number of possible combinations of c different variables, while the resulting kernel (κ^c_{mC}) is computed by

$$\begin{aligned}\kappa^c_{\text{mC}}(\mathbf{x},\mathbf{z}) &= \langle \phi^c_{\text{mC}}(\mathbf{x}), \phi^c_{\text{mC}}(\mathbf{z})\rangle \\ &= \sum_{\mathbf{b}\in\mathbb{B}_c} \mathbf{x}^{\mathbf{b}}\mathbf{z}^{\mathbf{b}} = \sum_{\mathbf{b}\in\mathbb{B}_c} (\mathbf{x}\odot\mathbf{z})^{\mathbf{b}} = \binom{\langle \mathbf{x},\mathbf{z}\rangle}{c} = \binom{\kappa_{\text{mL}}(\mathbf{x},\mathbf{z})}{c},\end{aligned} \tag{6.4}$$

where the last but one equality holds because the number of times that $\forall\, i, \mathbf{b}_i = 1$ such that $\mathbf{x}_i\mathbf{z}_i = 1$ is exactly the number of combination of c variables taken from $\langle \mathbf{x},\mathbf{z}\rangle$.

Figure 6.2 gives a graphical representation of the computation of the mC-kernel of degree 2.

Properties of the mC-kernel

The mC-kernel owns some peculiarities that are worth to mention:

- κ_{mC}^{c} is the ANOVA kernel (see Section 4.4) of degree c with binary vectors as input;
- the sparsity of the kernel increases with the increasing of the degree c. This is due to the fact that the number of active conjunctions decreases as the number of involved variables increases (see Figure 6.5);
- the dimension of the feature space is not monotonically increasing with c. The dimension reaches its maximum when $c = \lfloor \frac{n}{2} \rfloor$ and then starts to decrease, this is due to the binomial coefficient;
- when $c = 1$ the resulting mC-kernel is equal to the mL-kernel, i.e., the linear kernel:

$$\kappa_{\text{mC}}^{1}(\mathbf{x}, \mathbf{z}) = \binom{\langle \mathbf{x}, \mathbf{z} \rangle}{1} = \langle \mathbf{x}, \mathbf{z} \rangle = \kappa_{\text{mL}}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{LIN}}(\mathbf{x}, \mathbf{z});$$

- given a vector $\mathbf{x}$ such that $\|\mathbf{x}\| < c$, its mapping in the embedding space will be the null vector. From a computational stand point, this can cause issues, e.g., whenever the induced kernel has to be normalized;
- the computational complexity of this kernel function is very efficient $\mathcal{O}(n + c)$.

> **Proposition 6.1**
>
> Time complexity of κ_{mC}^{c} computation is $\mathcal{O}(n + c)$.

Proof. Given $\mathbf{x}, \mathbf{z} \in \{0, 1\}^n$, the complexity of the dot-product $\langle \mathbf{x}, \mathbf{z} \rangle$ is linear in n, so it is $\mathcal{O}(n)$. For any $q \in \mathbb{N}$, the binomial coefficient q choose c is computed by

$$\binom{q}{c} = \prod_{i=0}^{c-1} \frac{q - i}{i + 1},$$

and hence it has a complexity $\mathcal{O}(c)$. Therefore, the overall computational complexity is $\mathcal{O}(n + c)$. □

In practice, c is usually order of magnitude smaller than n, i.e., $c \ll n$, so the complexity of κ_{mC}^{c} is reduced to $\mathcal{O}(n)$.

Connection with the DNF kernel [Sad01]

As said previously, both works [KS05] and [Sad01] independently proposed the monotone DNF kernel, in which the feature space is represented by all possible conjunctions of positive Boolean literals.

This kernel is applicable to points in $\mathbb{R}^n$, however, if we restrict to the case in which the input vectors are in $\{0,1\}^n$ then it can be computed as follows:

$$\kappa_{\text{mDNF}}(\mathbf{x},\mathbf{z}) = -1 + \prod_{i=1}^{n}(\mathbf{x}_i\mathbf{z}_i + 1) = 2^{\langle \mathbf{x},\mathbf{z}\rangle} - 1.$$

Actually, the monotone form of the DNF kernel counts how many (any degree) conjunctions of variables are satisfied in both $\mathbf{x}$ and $\mathbf{z}$, and this is related to what the mC-kernel does. In fact, we can express the monotone DNF kernel as a linear combination of mC-kernels of degree $1 \leq c \leq n$ as in the following:

$$\kappa_{\text{mDNF}}(\mathbf{x},\mathbf{z}) = 2^{\langle \mathbf{x},\mathbf{z}\rangle} - 1 = \sum_{d=0}^{n}\binom{\langle \mathbf{x},\mathbf{z}\rangle}{c} - 1 = \sum_{c=1}^{n}\kappa^c_{\text{mC}}(\mathbf{x},\mathbf{z}).$$

6.3.3 Monotone Disjunctive kernel

The disjunction of two Boolean variables $x, z \in \{0,1\}$, denoted by $x \vee z$, is satisfied anytime at least one of the variables is *true*, or in other words, it is not satisfied if and only if $x = z = 0$.

The embedding of the monotone Disjunctive kernel (mD-kernel) is the same as the mC-kernel, but the logical interpretation is different. In the mD-kernel, as the name suggests, the combinations of variables inside the feature space represent disjunctions of variables, e.g., $x_1x_4x_6 \equiv x_1 \vee x_4 \vee x_6$. Formally, the embedding of the mD-kernel of degree $d \in [n]$ is given by

$$\phi^d_{\text{mD}} : \mathbf{x} \mapsto (\phi^{(\mathbf{b})}_{\text{mD}}(\mathbf{x}))_{\mathbf{b}\in\mathbb{B}_d}, \tag{6.5}$$

where

$$\phi^{(\mathbf{b})}_{\text{mD}}(\mathbf{x}) = [\![\langle \mathbf{x},\mathbf{b}\rangle > 0]\!], \tag{6.6}$$

and $\mathbb{B}_d$ is defined as in the previous section. As for the conjunctive case, the dimension of the embedding space is $\binom{n}{d}$. From a logical stand point, the mD-kernel of degree d between $\mathbf{x}$ and $\mathbf{z}$ counts the number of disjunctions of d variables that are satisfied in both $\mathbf{x}$ and $\mathbf{z}$.

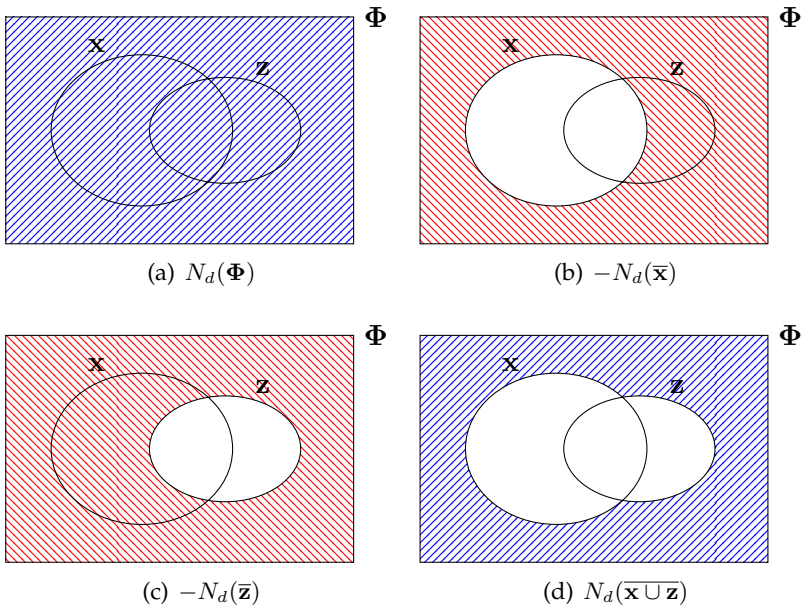
In order to ease the understanding of how the mD-kernel (κ^d_{mD}) is computed, we rely on a set theory interpretation. Let us denote with $N_d(\mathbf{x}) = \binom{|\mathbf{x}|}{d}$ the number of

combinations of size d that can be formed using elements of the set $\mathbf{x}$, and since $\mathbf{\Phi}$ (as defined in Section 6.2) is the universal set holds that $N_d(\mathbf{\Phi}) = \binom{n}{d}$.

To obtain the value $\kappa^d_{\text{mD}}(\mathbf{x}, \mathbf{z})$ we have to count all the d-disjunctions that contain at least an active variable of $\mathbf{x}$ and $\mathbf{z}$ (potentially the same). A more intuitive way to compute this value is to face the problem in a negative fashion, that is, we subtract from all the possible d-combinations the ones with no active elements from both $\mathbf{x}$ and $\mathbf{z}$.

First of all, from the total $N_d(\mathbf{\Phi})$ we subtract all the possible inactive combinations of $\mathbf{x}$, that is, using the set interpretation, $N_d(\overline{\mathbf{x}})$. Analogously, we can do the same for $\mathbf{z}$. Now, we have removed twice the combinations made of elements taken from $\overline{\mathbf{x} \cup \mathbf{z}}$ and thus we need to add its contribution once, that is $N_d(\overline{\mathbf{x} \cup \mathbf{z}})$. We can now formally define κ^d_{mD} as

$$\begin{aligned}\kappa^d_{\text{mD}}(\mathbf{x}, \mathbf{z}) &= \langle \phi^d_{\text{mD}}(\mathbf{x}), \phi^d_{\text{mD}}(\mathbf{z}) \rangle = N_d(\mathbf{\Phi}) - N_d(\overline{\mathbf{x}}) - N_d(\overline{\mathbf{z}}) + N_d(\overline{\mathbf{x} \cup \mathbf{z}}) \\ &= \binom{n}{d} - \binom{n - |\mathbf{x}|}{d} - \binom{n - |\mathbf{z}|}{d} + \binom{n - |\mathbf{x}| - |\mathbf{z}| + |\mathbf{x} \cap \mathbf{z}|}{d} \\ &= \binom{n}{d} - \binom{n - \langle \mathbf{x}, \mathbf{x} \rangle}{d} - \binom{n - \langle \mathbf{z}, \mathbf{z} \rangle}{d} + \binom{n - \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{z}, \mathbf{z} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle}{d}.\end{aligned} \tag{6.7}$$



(a) $N_d(\mathbf{\Phi})$ (b) $-N_d(\overline{\mathbf{x}})$

(c) $-N_d(\overline{\mathbf{z}})$ (d) $N_d(\overline{\mathbf{x} \cup \mathbf{z}})$

Figure 6.3: The blue striped sections show the set of elements, (a) and (d), used to create the *active* combinations, while the red ones, (b) and (c), are the combinations that are discarded from the counting.

Figure 6.3 gives a graphical intuition of how the kernel is computed. Figure 6.3(a)

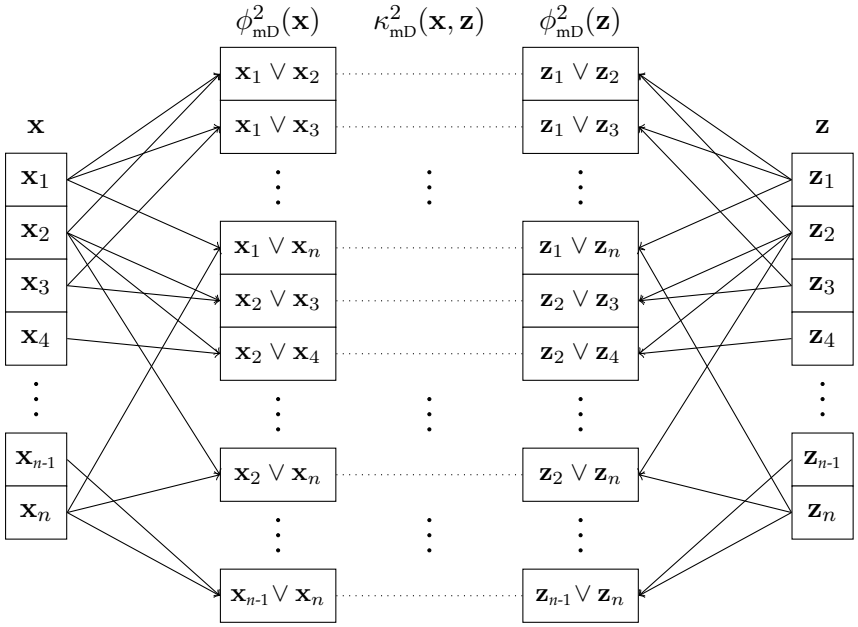


Figure 6.4: Depiction of the mD-kernel of degree 2: firstly the input vectors are mapped into the feature space which is formed by all disjunctions with arity 2 (without repetition). Then, the kernel is computed by "matching" (dotted lines) the corresponding features. Arrows from the input vectors to the feature vectors indicate when a variable influences the disjunction.

represents all the possible combinations while Figure 6.3(b) and Figure 6.3(c) show the set of elements used to create the combinations that are discarded from the counting. Finally, Figure 6.3(d) illustrates the set of variables that are used to re-add the combinations which have been discarded twice. Figure 6.4 instead gives a graphical representation of the computation of the mD-kernel of degree 2.

Properties of the mD-kernel

The mD-kernel owns the following nice properties:

- for any degree $d \geq 2$ the resulting mD-kernel matrix is fully dense.

> **Proposition 6.2**
>
> For any dataset $\mathcal{D}$ and for any $d \geq 2$ the mD-kernel matrix of degree d induced by $\mathcal{D}$, i.e., $\mathbf{K}_{i,j} = \kappa^d_{\text{mD}}(\mathbf{x}_i, \mathbf{x}_j)$, is fully dense.

Proof. Given two generic examples $\mathbf{x}$, $\mathbf{z}$ from the dataset $\mathcal{D}$ such that $\exists\ p, q \mid \mathbf{x}_p = 1 \wedge \mathbf{z}_q = 1$, then, for $d \geq 2$, the feature space induced by ϕ^d_{mD} has at least

one feature which contains the disjunction between the p-th and the q-th variables of the vectors, which is active in both $\phi^d_{\text{mD}}(\mathbf{x})$ and $\phi^d_{\text{mD}}(\mathbf{z})$ and hence the kernel $\kappa^d_{\text{mD}}(\mathbf{x}, \mathbf{z}) \geq 1$. □

- the mD-kernel has the same complexity as the mC-kernel.

> **Proposition 6.3**
>
> Time complexity of κ^d_{mD} computation is $\mathcal{O}(n + d)$;

Proof. As said in the proof of the Proposition 6.3.2, the complexity of $\binom{q}{d}$ computation is $\mathcal{O}(d)$, while the complexity of $|\mathbf{x}| := \|\mathbf{x}\|^2$, $|\mathbf{z}| := \|\mathbf{z}\|^2$ and $|\mathbf{x} \cap \mathbf{z}| := \langle \mathbf{x}, \mathbf{z} \rangle$ is $\mathcal{O}(n)$. So, with the exception of the first binomial which has a complexity of $\mathcal{O}(d)$, all the binomials in $\kappa^d_{\text{mD}}(\mathbf{x}, \mathbf{z})$ has a complexity of $\mathcal{O}(n + d)$ and hence the overall complexity is equals to $\mathcal{O}(n + d)$. □

Likewise the mC-kernel, in general, the degree d is order of magnitude smaller than n (i.e., $d \ll n$) and hence the complexity of κ^d_{mD} can be reduced to $\mathcal{O}(n)$.

- the dimension of the feature space is not monotonically increasing with d. The dimension reaches its maximum when $d = \lfloor \frac{n}{2} \rfloor$ and then starts to decrease, this is due to the binomial coefficient;

- as for the mC-kernel, when $d = 1$ the mD-kernel is equal to the linear one

$$\begin{aligned}
\kappa^1_{\text{mD}}(\mathbf{x}, \mathbf{z}) &= \binom{n}{1} - \binom{n - |\mathbf{x}|}{1} - \binom{n - |\mathbf{z}|}{1} + \binom{n - |\mathbf{x}| - |\mathbf{z}| + |\mathbf{x} \cap \mathbf{z}|}{1} \\
&= n - (n - |\mathbf{x}|) - (n - |\mathbf{z}|) + (n - |\mathbf{x}| - |\mathbf{z}| + |\mathbf{x} \cap \mathbf{z}|) \\
&= |\mathbf{x} \cap \mathbf{z}| := \langle \mathbf{x}, \mathbf{z} \rangle = \kappa_{\text{mL}}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{LIN}}(\mathbf{x}, \mathbf{z});
\end{aligned}$$

- given a vector $\mathbf{x}$ such that $\|\mathbf{x}\| \geq n - d$, its mapping in the embedding space will be the vector with all entries equal to 1, since in every combinations of d variables at least one is true.

6.4 Non-monotone Boolean kernels

Conversely to the monotone case, non-monotone Boolean formulas can contain negated literals, e.g., $\neg x_i$, and for this reason there is the need to consider the negated variables in the computation of the kernels.

6.4.1 Non-monotone Literal kernel

In order to include the contribution of the negated literals, we need to add to the mL-kernel the number of false literals in common between $\mathbf{x}$ and $\mathbf{z}$. Let us define the negation kernel first, which simply count how many false variables the two input vectors have in common. The embedding function of the negation simply maps the vectors onto their "opposite", that is

$$\phi_{\mathrm{N}}(\mathbf{x}) = \mathbf{1}_n - \mathbf{x}, \tag{6.8}$$

and consequently the negation kernel is defined as

$$\kappa_{\mathrm{N}}(\mathbf{x}, \mathbf{z}) = \langle \phi_{\mathrm{N}}(\mathbf{x}), \phi_{\mathrm{N}}(\mathbf{z}) \rangle = \langle (\mathbf{1}_n - \mathbf{x}), (\mathbf{1}_n - \mathbf{z}) \rangle = n - \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{z}, \mathbf{z} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle. \tag{6.9}$$

The Non-monotone Literal (L) kernel, $\kappa_{\mathrm{L}}(\mathbf{x}, \mathbf{z})$, counts how many true and false variables $\mathbf{x}$ and $\mathbf{z}$ have in common. Its feature mapping is simply the concatenation of the two mappings ϕ_{mL} and ϕ_{N}, and thus the kernel is defined as a sum of kernels as in the following:

$$\kappa_{\mathrm{L}}(\mathbf{x}, \mathbf{z}) = \kappa_{\mathrm{mL}}(\mathbf{x}, \mathbf{z}) + \kappa_{\mathrm{N}}(\mathbf{x}, \mathbf{z}) = n - \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{z}, \mathbf{z} \rangle + 2\langle \mathbf{x}, \mathbf{z} \rangle. \tag{6.10}$$

Clearly the embedding space has dimension $2n$, and for any input vector $\mathbf{x}$ holds that $\kappa_{\mathrm{L}}(\mathbf{x}, \mathbf{x}) = n$. The proof is trivial since every element of $\mathbf{x}$ is equal to itself no matter whether is true or false.

6.4.2 Non-monotone Conjunctive kernel

The non-monotone Conjunctive (C) kernel is related to the mC-kernel: it counts how many non-monotone conjunctions, of a certain arity c, are true in both $\mathbf{x}$ and $\mathbf{z}$. Conceptually is very similar to the monotone case but it has to consider also variables in their negated state.

A key aspect of the C-kernel is that its feature space contains all the possible combinations of the input variables, but each variable is in either its positive or negative state. However, we can observe that, anytime a variable compare in a conjunction in both its positive and negative states the conjunction can not be satisfied (i.e., it is a contradiction). We can leverage on this observation and define the C-kernel as a special case of the mC-kernel. Actually the C-kernel of degree $c \in [n]$ is the mC-kernel applied to vectors in the space spanned by the embedding function of the L-kernel, that is:

$$\kappa_{\mathrm{C}}^{c}(\mathbf{x}, \mathbf{z}) = \binom{\langle \phi_{\mathrm{L}}(\mathbf{x}), \phi_{\mathrm{L}}(\mathbf{z}) \rangle}{c} = \binom{\kappa_{\mathrm{L}}(\mathbf{x}, \mathbf{z})}{c} = \binom{n - \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{z}, \mathbf{z} \rangle + 2\langle \mathbf{x}, \mathbf{z} \rangle}{c}, \tag{6.11}$$

which is by definition a kernel. From the definition above we can argue that the feature space has dimension $\binom{2n}{c}$. Nevertheless, for the observation made before, we can also define a valid feature space of dimension $2^c\binom{n}{c}$ that is the number of possible conjunctions of c literals taken in either positive or negative state. All the combinations (i.e., features) which are not considered, that are $\binom{2n}{c} - 2^c\binom{n}{c}$, are always 0 because they form inconsistent formulas, i.e., $x_1 \wedge \neg x_1$, and thus they can be omitted.

Another nice property of the C-kernel is the following: for any $c \in [n]$ and any input vector $\mathbf{x}$ holds that $\kappa_{\text{C}}^c(\mathbf{x}, \mathbf{x}) = \binom{n}{c}$. The proof is straightforward since $\forall\, \mathbf{x}, \kappa_{\text{L}}(\mathbf{x}, \mathbf{x}) = n$.

6.4.3 Non-monotone Disjunctive kernel

The non-monotone Disjunctive (D) kernel counts how many non-monotone disjunctions, of a certain arity d, are true in both $\mathbf{x}$ and $\mathbf{z}$. Clearly, the embedding is the same as the C-kernel, but the logical interpretation is different since here we treat disjunctions instead of conjunctions. As for the monotone case, we derive the kernel function $\kappa_{\text{D}}^d(\mathbf{x}, \mathbf{z})$ in a negative fashion.

The number of all possible combinations of arity d of Boolean variables that can be in their negated form is $2^d\binom{n}{d}$, that is the dimension of the feature space. For both, $\mathbf{x}$ and $\mathbf{z}$ we have to discard the combinations that are false, which are exactly $\binom{n}{d}$ because for each set of d different variables, there exists only one assignment of the negations such that the disjunction is falsified. For example, let us take the variables $x_1 = 1, x_2 = 0$ and $x_3 = 1$, only the disjunction $\neg x_1 \vee x_2 \vee \neg x_3$ is false, while all the others $2^3 - 1$ negation assignments satisfy the disjunction. Finally, we have to re-add the false combinations that have been discarded twice, that are the combinations made with variables that are false in both vectors. This can be seen as the opposite of what the C-kernel computes, however since we generate all possible combinations with all possible negation assignments, the counting is actually the same as the C-kernel. This holds because it is like to consider the C-kernel but with both vectors negated point-wise. We can thus define the D-kernel of degree $d \in [n]$ as

$$\kappa_{\text{D}}^d(\mathbf{x}, \mathbf{z}) = 2^d\binom{n}{d} - \binom{n}{d} - \binom{n}{d} + \kappa_{\text{C}}^c(\mathbf{x}, \mathbf{z}) = (2^d - 2)\binom{n}{d} + \kappa_{\text{C}}^c(\mathbf{x}, \mathbf{z}). \tag{6.12}$$

Similarly to the C-kernel, the D-kernel owns the following property: for any $d \in [n]$ input vector $\mathbf{x}$ holds that $\kappa_{\text{D}}^d(\mathbf{x}, \mathbf{x}) = (2^d - 1)\binom{n}{d}$. The proof is trivial since $\forall\, \mathbf{x}, \kappa_{\text{C}}^d(\mathbf{x}, \mathbf{x}) = \binom{n}{d}$.

6.5 Boolean kernels computation

The computational complexity of the Boolean kernels described in the previous sections is bounded by the complexity of the calculation of the binomial coefficient, that is $\mathcal{O}(k)$

with k the arity of the combinations. Hence, computing an entire kernel matrix over n-dimensional examples taken from a dataset with l examples, would lead to a complexity of $\mathcal{O}((k+n)\cdot l^2)$. Despite the fact that it is not possible to reduce such complexity, we can take advantage of the recursive nature of the binomial coefficient in order to compute the kernels in a recursive fashion. By doing so, it is possible to compute higher degree kernel matrices by recovering kernel matrices of lower degrees.

Let the matrix $\mathbf{K}^0$ be the base (Boolean) kernel matrix over the dataset $\mathcal{D}$, such that $\mathbf{K}^0_{i,j} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, for $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ and some ϕ with codomain $\{0,1\}^n$. Then, we can recursively define the Boolean kernels for both monotone and the non-monotone case as described in the following.

6.5.1 mC-kernel

By definition, the mC-kernel matrix of arity 1, that is $\mathbf{K}^1_{\text{mC}}$, is equivalent to the base kernel matrix $\mathbf{K}^0$. By using $\mathbf{K}^0$ as base case, we can recursively define the mC-kernel matrix as

$$\mathbf{K}^{c+1}_{\text{mC}} = \mathbf{K}^{c}_{\text{mC}} \odot \left(\frac{1}{c+1} \left(\mathbf{K}^{1}_{\text{mC}} - c\mathbf{1}_l\mathbf{1}_l^\intercal \right) \right).$$

6.5.2 mD-kernel

Let us define the matrices

$$\mathbf{S} = \text{diag}(\mathbf{K}^0) \cdot \mathbf{1}_l^\intercal, \quad \mathbf{N}_x = n\mathbf{1}_l\mathbf{1}_l^\intercal - \mathbf{S}, \text{ and } \mathbf{N}_{xz} = \mathbf{N}_x - \mathbf{S}^\intercal + \mathbf{K}^0.$$

Then we define, recursively in its parts, the mD-kernel matrix $\mathbf{K}^d_{\text{mD}}$ as

$$\mathbf{K}^d_{\text{mD}} = \mathbf{N}^{(d)} - \mathbf{N}^{(d)}_x - \mathbf{N}^{(d)\intercal}_x + \mathbf{N}^{(d)}_{xz},$$

where

$$\mathbf{N}^{(d+1)} = \mathbf{N}^{(d)} \odot \left(\frac{n-d}{d+1} \mathbf{1}_l\mathbf{1}_l^\intercal \right),$$
$$\mathbf{N}^{(d+1)}_x = \mathbf{N}^{(d)}_x \odot \left(\frac{1}{d+1} \left(\mathbf{N}_x - d\mathbf{1}_l\mathbf{1}_l^\intercal \right) \right), \text{ and}$$
$$\mathbf{N}^{(d+1)}_{xz} = \mathbf{N}^{(d)}_{xz} \odot \left(\frac{1}{d+1} \left(\mathbf{N}_{xz} - d\mathbf{1}_l\mathbf{1}_l^\intercal \right) \right),$$

with the corresponding base cases $\mathbf{N}^{(1)} = n\mathbf{1}_l\mathbf{1}_l^\intercal$, $\mathbf{N}^{(1)}_x = \mathbf{N}_x$ and $\mathbf{N}^{(1)}_{xz} = \mathbf{N}_{xz}$.

6.5.3 C-kernel

By relying on the previous definition of $\mathbf{S}$ and the base case kernel matrix

$$\mathbf{K}_{\mathrm{C}}^{1} = n\mathbf{1}_l\mathbf{1}_l^{\intercal} - \mathbf{S} - \mathbf{S}^{\intercal} + 2\mathbf{K}^{0},$$

the C-kernel matrix can be recursively defined by

$$\mathbf{K}_{\mathrm{C}}^{c+1} = \mathbf{K}_{\mathrm{C}}^{c} \odot \left(\frac{1}{c+1} \left(\mathbf{K}_{\mathrm{C}}^{1} - c\mathbf{1}_l\mathbf{1}_l^{\intercal} \right) \right).$$

6.5.4 D-kernel

Using the previous definitions of the matrices $\mathbf{N}^{(d)}$ and $\mathbf{K}_{\mathrm{C}}^{c}$, we can define the D-kernel matrix, recursively in its parts, as

$$\mathbf{K}_{\mathrm{D}}^{d+1} = \left((2^d - 2)\mathbf{1}_l\mathbf{1}_l^{\intercal} \right) \odot \mathbf{N}^{(d)} - \mathbf{K}_{\mathrm{C}}^{d}.$$

6.6 Combination of Boolean kernels

Given the family of Boolean kernels defined in the previous sections, we have now all the basic elements to build more complex Boolean kernels that represent a specific logical concept. Table 6.1 provides a summary of all the just presented Boolean kernels. It easy to see that all kernels are functions of dot-products of the input vectors, and this allow us to define new kernels by replacing those dot-products with other Boolean kernels. The logical interpretation of the new kernel depends on how the base Boolean kernels are combined. In the following we present some new Boolean kernels generated by using this method.

6.6.1 Disjunctive Normal Form kernels

In Section 4.4 we described the DNF-kernel and the mDNF-kernel [Sad01]. These kernels owe their names to the fact that the solution of the kernel machine can be interpreted as a (monotone) DNF. However, the way we have assigned names to kernels is strictly related to which feature space they create. For this reason, we are going to re-use the same kernel names for two new Boolean kernels in which the feature space is formed by DNF formulas. In the remainder, when not specified differently, we will refer with DNF-kernel and mDNF-kernel to the following kernels.

κ	$\kappa(\mathbf{x}, \mathbf{z})$	$\kappa(\mathbf{x}, \mathbf{x})$	$\lvert\phi\rvert$
κ_{mL}	$\langle \mathbf{x}, \mathbf{z} \rangle$	$\lVert \mathbf{x} \rVert$	n
κ^c_{mC}	$\binom{\kappa_{\text{mL}}(\mathbf{x}, \mathbf{z})}{c}$	$\binom{\lvert\mathbf{x}\rvert}{c}$	$\binom{n}{c}$
κ^d_{mD}	$\binom{n}{d} - \binom{n - \lVert\mathbf{x}\rVert}{d} - \binom{n - \lVert\mathbf{z}\rVert}{d} + \binom{n - \lVert\mathbf{x}\rVert - \lVert\mathbf{z}\rVert + \langle \mathbf{x}, \mathbf{z} \rangle}{d}$	$\binom{n}{d} - \binom{n - \lVert\mathbf{x}\rVert}{d}$	$\binom{n}{d}$
κ_{N}	$n - \lVert\mathbf{x}\rVert - \lVert\mathbf{z}\rVert + \langle \mathbf{x}, \mathbf{z} \rangle$	$n - \lVert\mathbf{x}\rVert$	n
κ_{L}	$\kappa_{\text{mL}}(\mathbf{x}, \mathbf{z}) + \kappa_{\text{N}}(\mathbf{x}, \mathbf{z})$	n	$2n$
κ^c_{C}	$\binom{\kappa_{\text{L}}(\mathbf{x}, \mathbf{z})}{c}$	$\binom{n}{c}$	$2^c\binom{n}{c}$
κ^d_{D}	$(2^d - 2)\binom{n}{d} + \kappa^d_{\text{C}}(\mathbf{x}, \mathbf{z})$	$(2^d - 1)\binom{n}{d}$	$2^d\binom{n}{d}$

Table 6.1: Summary of the just described Boolean kernels: $|\phi|$ stands for the dimension of the feature space.

Monotone DNF-kernel

We have already given a definition of Disjunctive Normal Form in Section 4.4. Here we focus on the monotone case. The idea of the monotone DNF kernel (mDNF-kernel) is to compute the dot-product of vectors in a feature space composed by monotone DNF (mDNF) formulas over the input variables.

In particular, the variables are mapped onto a space containing all the monotone DNF formulas composed by disjunctions of exactly d conjunctive clauses formed by c literals, that we will call mDNF(d,c) for brevity. For example, given $d = 2$ and $c = 3$ a possible mDNF is $(x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$.

Since, DNFs are disjunctions of conjunctive clauses we can combine the embedding maps of the mC-kernel and the mD-kernel in this way

$$\phi^{d,c}_{\text{mDNF}} : \mathbf{x} \mapsto \phi^d_{\text{mD}}(\phi^c_{\text{mC}}(\mathbf{x})) \tag{6.13}$$

obtaining the desired feature space for the mDNF(d,c).

From the embedding definition, we can see that in order to calculate the mDNF-kernel we have to compute the mD-kernel of degree d in the space formed by all conjunctions of degree c.

Using the same sets analogy as in Section 6.3.3, the mDNF-kernel between the vec-

tors $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$ is calculated by:

$$\begin{aligned}\kappa^{d,c}_{\text{mDNF}}(\mathbf{x},\mathbf{z}) &= \binom{N_c(\mathbf{\Phi})}{d} - \binom{N_c(\mathbf{\Phi}) - N_c(\mathbf{x})}{d} - \binom{N_c(\mathbf{\Phi}) - N_c(\mathbf{z})}{d} \\ &\qquad + \binom{N_c(\mathbf{\Phi}) - N_c(\mathbf{x}) - N_c(\mathbf{z}) + N_c(\mathbf{x}\cap\mathbf{z})}{d} \\ &= \binom{\binom{n}{c}}{d} - \binom{\binom{n}{c} - \kappa^c_{\text{mC}}(\mathbf{x},\mathbf{x})}{d} - \binom{\binom{n}{c} - \kappa^c_{\text{mC}}(\mathbf{z},\mathbf{z})}{d} \\ &\qquad + \binom{\binom{n}{c} - \kappa^c_{\text{mC}}(\mathbf{x},\mathbf{x}) - \kappa^c_{\text{mC}}(\mathbf{z},\mathbf{z}) + \kappa^c_{\text{mC}}(\mathbf{x},\mathbf{z})}{d}.\end{aligned} \tag{6.14}$$

Note that in order to compute the kernel we need to know the dimension of the feature space of the mC-kernel, that is $\binom{n}{c}$. By construction, the dimension of the feature space is $\binom{\binom{n}{c}}{d}$.

As for the previous monotone kernels, it is easy to see that for $d = c = 1$ the mDNF-kernel is the linear kernel:

$$\begin{aligned}\kappa^{1,1}_{\text{mDNF}}(\mathbf{x},\mathbf{z}) &= n - (n - |\mathbf{x}|) - (n - |\mathbf{z}|) + (n - |\mathbf{x}| - |\mathbf{z}| + |\mathbf{x}\cap\mathbf{z}|) \\ &= \kappa^1_{\text{mD}}(\mathbf{x},\mathbf{z}) = \kappa_{\text{LIN}}(\mathbf{x},\mathbf{z}).\end{aligned}$$

Notation overload

We are reusing for these new kernels the name (m)DNF-kernel which has been presented in Section 4.4. However, we think that such name is the right one for our kernels since they actually represent what is inside the feature space, and hence, in order to be consistent, we borrowed the names.

Non-monotone DNF-kernel

The difference between a monotone DNF and a non-monotone one is simply the fact that the latter can contain negated variables inside the conjunctive clauses. So, from an embedding point of view, the only difference is the inner embedding, which is in the DNF case the mapping function of the C-kernel. Formally, the embedding function of the DNF-kernel with d conjunctive clauses made of c literals, i.e., DNF-kernel(d,c), is

$$\phi^{d,c}_{\text{DNF}} : \mathbf{x} \mapsto \phi^d_{\text{mD}}(\phi^c_{\text{C}}(\mathbf{x})). \tag{6.15}$$

With the same approach used in the monotone DNF we can define the DNF-kernel(d,c) as in the following:

$$\kappa^{d,c}_{\text{DNF}}(\mathbf{x},\mathbf{z}) = \binom{2^c\binom{n}{c}}{d} - \binom{2^c\binom{n}{c} - \kappa^c_{\text{C}}(\mathbf{x},\mathbf{x})}{d} - \binom{2^c\binom{n}{c} - \kappa^c_{\text{C}}(\mathbf{z},\mathbf{z})}{d} + \binom{2^c\binom{n}{c} - \kappa^c_{\text{C}}(\mathbf{x},\mathbf{x}) - \kappa^c_{\text{C}}(\mathbf{z},\mathbf{z}) + \kappa^c_{\text{C}}(\mathbf{x},\mathbf{z})}{d}. \tag{6.16}$$

By construction, the dimension of the feature space is $\binom{2^c\binom{n}{c}}{d}$. Similarly to the monotone DNF, it is easy to see that for $d = c = 1$ the DNF-kernel is the (non-monotone) Literal kernel:

$$\kappa^{1,1}_{\text{DNF}}(\mathbf{x},\mathbf{z}) = n - (n-n) - (n-n) + (n-n-n+\kappa_{\text{L}}(\mathbf{x},\mathbf{z})) = \kappa_{\text{L}}(\mathbf{x},\mathbf{z}).$$

6.6.2 Conjunctive Normal Form kernels

> **Conjunctive Normal Form**
>
> In logic, a Conjunctive Normal Form (CNF) is a normalization of a logical formula which is a conjunction of disjunctive clauses, for example, given the Boolean variables $x_1, \dots x_4 \in \{0,1\}$, $(x_1 \vee x_2 \vee x_3) \wedge x_2 \wedge (x_1 \vee x_4)$ is a valid CNF formula.
>
> In particular, the example above is a monotone CNF. A non-monotone CNF, or simply CNF, can contain negated variables, e.g., $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge x_2 \wedge (\neg x_1 \vee x_4)$.

Monotone CNF-kernel

The construction of a CNF formula is very close to the DNF one, the only difference is that the operations (i.e., or/and) are applied in the reverse order. For this reason, to build the embedding of the CNF-kernel it is sufficient to swap the combination of the embedding functions w.r.t. to the DNF case. By using the same notation as for the DNF, c indicates the number of conjunctions and d the cardinality of the disjunctive clauses. The monotone CNF embedding function is given by

$$\phi^{c,d}_{\text{mCNF}} : \mathbf{x} \mapsto \phi^c_{\text{mC}}(\phi^d_{\text{mD}}(\mathbf{x})), \tag{6.17}$$

which leads to a feature space of dimension $\binom{\binom{n}{d}}{c}$. Consequently, the mCNF-kernel(c,d) is defined as

$$\kappa^{c,d}_{\text{mCNF}}(\mathbf{x},\mathbf{z}) = \binom{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{c}. \tag{6.18}$$

Evidently, when $d = c = 1$ the mCNF-kernel is the linear kernel:

$$\kappa^{1,1}_{\text{mCNF}}(\mathbf{x}, \mathbf{z}) = \binom{\kappa^{1}_{\text{mD}}(\mathbf{x}, \mathbf{z})}{1} = \kappa^{1}_{\text{mD}}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{mL}}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{LIN}}(\mathbf{x}, \mathbf{z}).$$

Non-monotone CNF-kernel

For the same considerations made in the transition from the mDNF to the DNF-kernel, we can define the embedding function of the non-monotone CNF-kernel(c,d) as

$$\phi^{c,d}_{\text{CNF}} : \mathbf{x} \mapsto \phi^{c}_{\text{mC}}(\phi^{d}_{\text{D}}(\mathbf{x})) \tag{6.19}$$

and the corresponding kernel function as

$$\kappa^{c,d}_{\text{CNF}}(\mathbf{x}, \mathbf{z}) = \binom{\kappa^{d}_{\text{D}}(\mathbf{x}, \mathbf{z})}{c}. \tag{6.20}$$

Clearly the feature space has dimension $\binom{2^d\binom{n}{d}}{c}$ and if $d = c = 1$ we have that

$$\kappa^{1,1}_{\text{CNF}}(\mathbf{x}, \mathbf{z}) = \binom{\kappa^{1}_{\text{D}}(\mathbf{x}, \mathbf{z})}{1} = \kappa^{1}_{\text{D}}(\mathbf{x}, \mathbf{z}) = \kappa_{\text{L}}(\mathbf{x}, \mathbf{z}).$$

6.7 Analysis of the expressiveness

In this section, we show some properties about the expressiveness of the just described kernels according to the spectral ratio of their corresponding kernel matrices (see Section 4.6).

We consider normalized kernels by means of the formula (4.6). The use of normalized kernels leads to a nice property about the relation of the SR of two kernel functions. Formally, let us define the following lemma.

Lemma 6.1

Let be $\widetilde{\kappa}$ and $\widetilde{\kappa}'$ two normalized kernel functions with their corresponding kernel matrices $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{K}}' \in \mathbb{R}^{m \times m}$. If, for any $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$, $\widetilde{\kappa}(\mathbf{x}, \mathbf{z})^2 \geq \widetilde{\kappa}'(\mathbf{x}, \mathbf{z})^2$ holds, then $\mathcal{C}(\widetilde{\mathbf{K}}) \leq \mathcal{C}(\widetilde{\mathbf{K}}')$.

Proof. Assume that $\widetilde{\kappa}(\mathbf{x}, \mathbf{z})^2 \geq \widetilde{\kappa}'(\mathbf{x}, \mathbf{z})^2$ holds for any $\mathbf{x}$, $\mathbf{z}$. Then, we can say that the

following inequality holds:

$$\|\widetilde{\mathbf{K}}\|_{\mathrm{F}}^2 = \sum_{i=1}^{m}\sum_{j=1}^{m}\widetilde{\mathbf{K}}_{i,j}^2 \geq \sum_{i=1}^{m}\sum_{j=1}^{m}\widetilde{\mathbf{K}}_{i,j}'^2 = \|\widetilde{\mathbf{K}}'\|_{\mathrm{F}}^2. \tag{6.21}$$

Since kernels are normalized, $\|\widetilde{\mathbf{K}}\|_{\mathrm{T}} = \|\widetilde{\mathbf{K}}'\|_{\mathrm{T}} = m$, we have that:

$$\mathcal{C}(\widetilde{\mathbf{K}}) = \frac{\|\widetilde{\mathbf{K}}\|_{\mathrm{T}}}{\|\widetilde{\mathbf{K}}\|_{\mathrm{F}}} = \frac{m}{\|\widetilde{\mathbf{K}}\|_{\mathrm{F}}} \leq \frac{m}{\|\widetilde{\mathbf{K}}'\|_{\mathrm{F}}} = \frac{\|\widetilde{\mathbf{K}}'\|_{\mathrm{T}}}{\|\widetilde{\mathbf{K}}'\|_{\mathrm{F}}} = \mathcal{C}(\widetilde{\mathbf{K}}').$$

□

6.7.1 Expressiveness of the mC-kernel

By construction, the features of the mC-kernel of degree c have a clear dependence with the features of the same kernel of degree $c-1$. For example, consider the feature $x_1x_2x_3$ (i.e., $x_1 \wedge x_2 \wedge x_3$) from the feature space of a mC-kernel of degree 3, its value is strictly related to the ones of the features x_1x_2, x_1x_3 and x_2x_3 of an mC-kernel of degree 2. Specifically, the conjunctive feature $x_1x_2x_3$ is true if and only if all the "sub-features" x_1x_2, x_1x_3 and x_2x_3 are also true, which, in turn, are true if and only if x_1, x_2 and x_3 are true. Figure 6.5 gives a visual representation of these dependencies.

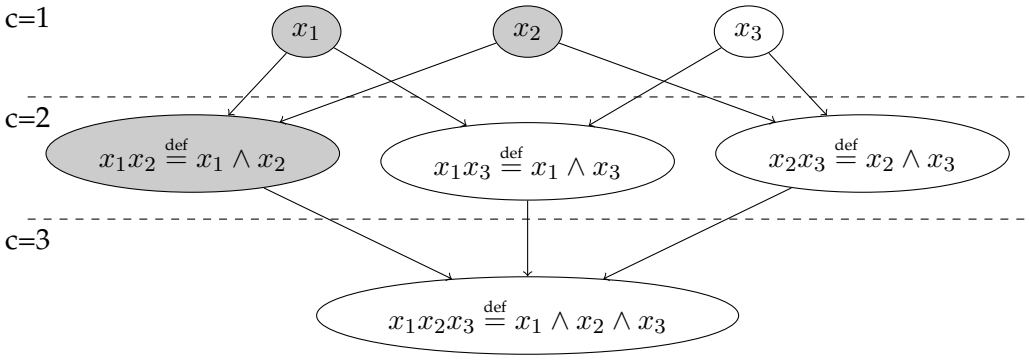


Figure 6.5: Dependencies between different degrees of the mC-kernel. Nodes are features and the grey ones are the active (i.e., true) features.

Intuitively, higher the arity of the conjunction less likely is that it is satisfied, and hence we expect that the higher the order of the mC-kernel, the higher the degree of sparsity of the corresponding kernel matrix. We will prove this is true (for the normalized kernel) and we also show that the degree c induces an order of expressiveness in $\widetilde{\kappa}_{\mathrm{mC}}^{c}(\mathbf{x}, \mathbf{z})$.

Theorem 6.1

Given $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$, $\|\mathbf{x}\|, \|\mathbf{z}\| \geq m$ for any choice of $c \in [m]$, the c-degree normalized mC-kernel $\widetilde{\kappa}^c_{\text{mC}}$ is more general than $(c+1)$-degree normalized mC-kernel $\widetilde{\kappa}^{c+1}_{\text{mC}}$, that is $\widetilde{\kappa}^c_{\text{mC}} \geq_G \widetilde{\kappa}^{c+1}_{\text{mC}}$.

Proof. By Lemma 6.7 it is sufficient to prove that

$$\widetilde{\kappa}^c_{\text{mC}}(\mathbf{x}, \mathbf{z})^2 \geq \widetilde{\kappa}^{c+1}_{\text{mC}}(\mathbf{x}, \mathbf{z})^2$$

is true. Let us call $n_x = \|\mathbf{x}\|$, $n_z = \|\mathbf{z}\|$ and $n_{xz} = \langle \mathbf{x}, \mathbf{z} \rangle$, then:

$$\widetilde{\kappa}^c_{\text{mC}}(\mathbf{x}, \mathbf{z})^2 = \frac{\kappa^c_{\text{mC}}(\mathbf{x}, \mathbf{z})^2}{\kappa^c_{\text{mC}}(\mathbf{x}, \mathbf{x})\kappa^c_{\text{mC}}(\mathbf{z}, \mathbf{z})} = \frac{\binom{\langle \mathbf{x}, \mathbf{z} \rangle}{c}^2}{\binom{\langle \mathbf{x}, \mathbf{x} \rangle}{c}\binom{\langle \mathbf{z}, \mathbf{z} \rangle}{c}} = \frac{\binom{n_{xz}}{c}}{\binom{n_x}{c}} \frac{\binom{n_{xz}}{c}}{\binom{n_z}{c}}.$$

Now, let us examine the first term of the product. We have:

$$\frac{\binom{n_{xz}}{c}}{\binom{n_x}{c}} = \frac{\frac{n_{xz}!}{c!(n_{xz}-c)!}}{\frac{n_x!}{c!(n_x-c)!}} = \frac{n_{xz}!}{n_x!} \frac{(n_x-c)!}{(n_{xz}-c)!}.$$

Since $n_{xz} \leq n_x$ by definition, we can say that:

$$\frac{\binom{n_{xz}}{c}}{\binom{n_x}{c}} = \frac{n_{xz}!}{n_x!} \frac{(n_x-c)!}{(n_{xz}-c)!} = \frac{n_{xz}!}{n_x!} \frac{(n_x-c)(n_x-c-1)\dots(n_{xz}-c+1)(n_{xz}-c)!}{(n_{xz}-c)!},$$

and so:

$$\frac{\binom{n_{xz}}{c}}{\binom{n_x}{c}} = \frac{n_{xz}!}{n_x!}[(n_x-c)(n_x-c-1)\dots(n_{xz}-c+1)] \geq$$
$$\frac{n_{xz}!}{n_x!}[(n_x-c-1)(n_x-c-2)\dots(n_{xz}-c)] = \frac{\binom{n_{xz}}{c+1}}{\binom{n_x}{c+1}}.$$

The same inequality is clearly valid for the other term $\binom{n_{xz}}{c}\binom{n_z}{c}^{-1}$, and hence:

$$\widetilde{\kappa}^c_{\text{mC}}(\mathbf{x}, \mathbf{z})^2 = \frac{\binom{n_{xz}}{c}}{\binom{n_x}{c}} \frac{\binom{n_{xz}}{c}}{\binom{n_z}{c}} \geq \frac{\binom{n_{xz}}{c+1}}{\binom{n_x}{c+1}} \frac{\binom{n_{xz}}{c+1}}{\binom{n_z}{c+1}} = \widetilde{\kappa}^{c+1}_{\text{mC}}(\mathbf{x}, \mathbf{z})^2.$$

□

Note that analogous analysis can be done for the non-monotone conjunctive kernel.

6.7.2 Expressiveness of the mD-kernel

With very similar considerations as for the mC-kernel, it is evident that there exists a dependence between features of a mD-kernel of degree d and $d+1$. Let us consider the feature $x_1x_2x_3$ (i.e., $x_1 \vee x_2 \vee x_3$) of the mD-kernel of degree 3. This will be active anytime at least one of the features x_1x_2, x_2x_3 or x_1x_3 of the mD-kernel of degree 2 is active. In Figure 6.6 a visual depiction of these dependencies is provided.

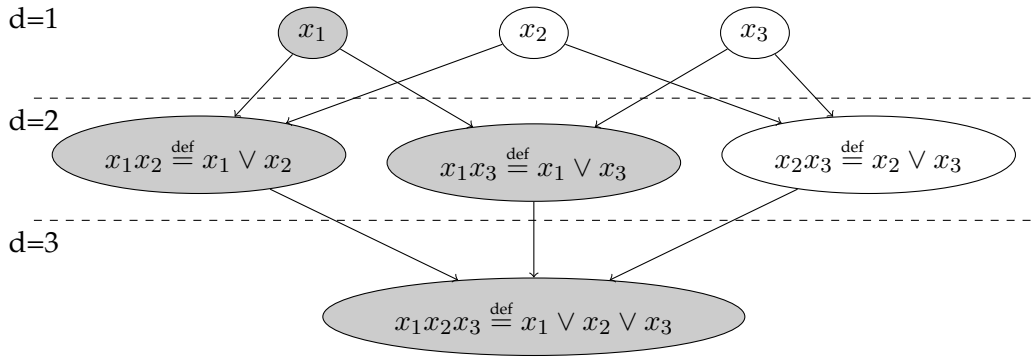


Figure 6.6: Dependencies between different degrees of the D-Kernel. The nodes are features and the colored ones are active features.

In this case, conversely to the mC-kernel, we expect that the higher the order of the mD-kernel the higher the degree of density of the corresponding kernel matrix. The following theorem proves that this holds.

Theorem 6.2

Given $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$, for any choice of $d \in \mathbb{N}$, the d-degree normalized mD-kernel $\widetilde{\kappa}^d_{\text{mD}}$ is more specific than the $(d+1)$-degree normalized mD-kernel $\widetilde{\kappa}^{d+1}_{\text{mD}}$, that is $\widetilde{\kappa}^d_{\text{mD}} \leq_G \widetilde{\kappa}^{d+1}_{\text{mD}}$.

Proof. By Lemma 6.7 it is sufficient to prove that

$$\widetilde{\kappa}^d_{\text{mD}}(\mathbf{x},\mathbf{z})^2 \leq \widetilde{\kappa}^{d+1}_{\text{mD}}(\mathbf{x},\mathbf{z})^2 \tag{6.22}$$

is true. Let us consider the left hand side term of the inequality (6.22):

$$\widetilde{\kappa}^d_{\text{mD}}(\mathbf{x},\mathbf{z})^2 = \frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})^2}{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{x})\kappa^d_{\text{mD}}(\mathbf{z},\mathbf{z})} = \frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{x})}\frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{z},\mathbf{z})}.$$

We can rewrite the term $\frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{x})}$ as:

$$\frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{x})} = \frac{N_d(\mathbf{\Phi}) - N_d(\overline{\mathbf{x}}) - N_d(\overline{\mathbf{z}}) + N_d(\overline{\mathbf{x}\cup\mathbf{z}})}{N_d(\mathbf{\Phi}) - N_d(\overline{\mathbf{x}})} = 1 - \frac{N_d(\overline{\mathbf{z}}) - N_d(\overline{\mathbf{x}\cup\mathbf{z}})}{N_d(\mathbf{\Phi}) - N_d(\overline{\mathbf{x}})}.$$

Let us explain the meaning of the last fraction: at the numerator, $N_d(\overline{\mathbf{z}}) - N_d(\overline{\mathbf{x}\cup\mathbf{z}})$ counts the number of d-combinations without repetition which have at least one active variable in $\mathbf{x}$ but not in $\mathbf{z}$. Similarly, the denominator, that is $N_d(\mathbf{\Phi}) - N_d(\overline{\mathbf{x}})$, counts the d-combinations without repetition which have at least one active variable in $\mathbf{x}$. Formally, we have:

$$\begin{aligned} N_d(\overline{\mathbf{z}}) - N_d(\overline{\mathbf{x}\cup\mathbf{z}}) &= (n_x - n_{xz})N_{d-1}(\overline{\mathbf{z}} \setminus \{a_z\}), \text{ and} \\ N_d(\mathbf{\Phi}) - N_d(\overline{\mathbf{x}}) &= n_x N_{d-1}(\mathbf{\Phi} \setminus \{a_x\}), \end{aligned}$$

where $a_x \in \mathbf{x}$ and $a_z \in \mathbf{z}$ are generic active variables in $\mathbf{x}$ and $\mathbf{z}$, respectively. So, we have that

$$\begin{aligned} \frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{x})} &= 1 - \frac{(n_x - n_{xz})}{n_x}\frac{N_{d-1}(\overline{\mathbf{z}} \setminus \{a_z\})}{N_{d-1}(\mathbf{\Phi} \setminus \{a_x\})} = 1 - \frac{(n_x - n_{xz})}{n_x}\frac{\binom{n-n_z-1}{d-1}}{\binom{n-1}{d-1}} \\ &= 1 - \frac{(n_x - n_{xz})}{n_x}\frac{\frac{(n-n_z-1)!}{(d-1)!(n-n_z-d)!}}{\frac{(n-1)!}{(d-1)!(n-d)!}} \\ &= 1 - \frac{(n_x - n_{xz})}{n_x}\frac{(n-n_z-1)!}{(n-1)!}\frac{(n-d)!}{(n-n_z-d)!} \\ &= 1 - \beta\frac{(n-d)(n-d-1)\dots(n-n_z-d+1)(n-n_z-d)!}{(n-n_z-d)!} \\ &= 1 - \beta[(n-d)(n-d-1)\dots(n-n_z-d+1)], \end{aligned}$$

where $\beta = \frac{(n_x - n_{xz})}{n_x}\frac{(n-n_z-1)!}{(n-1)!}$. Thus, it is easy to see that

$$\begin{aligned} &\frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{x})} = 1 - \beta[(n-d)(n-d-1)\dots(n-n_z-d+1)] \leq \\ &1 - \beta[(n-d-1)(n-d-2)\dots(n-n_z-d)] = \frac{\kappa^{d+1}_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^{d+1}_{\text{mD}}(\mathbf{x},\mathbf{x})}. \end{aligned}$$

Since the same consideration can be made for $\frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{z},\mathbf{z})}$, we can conclude that

$$\widetilde{\kappa}^d_{\text{mD}}(\mathbf{x},\mathbf{z})^2 = \frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{x})}\frac{\kappa^d_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^d_{\text{mD}}(\mathbf{z},\mathbf{z})} \leq \frac{\kappa^{d+1}_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^{d+1}_{\text{mD}}(\mathbf{x},\mathbf{x})}\frac{\kappa^{d+1}_{\text{mD}}(\mathbf{x},\mathbf{z})}{\kappa^{d+1}_{\text{mD}}(\mathbf{z},\mathbf{z})} = \widetilde{\kappa}^{d+1}_{\text{mD}}(\mathbf{x},\mathbf{z})^2.$$

□

Note that analogous considerations can be done for the non-monotone disjunctive kernel.

6.7.3 Expressiveness of the normal form kernels

In the normal form case, it is very difficult to define a full order of expressiveness in the lattice of the kernels. Nevertheless, there are some relations that we can be inherited from the (m)C-kernel and (m)D-kernel. In fact, given a normal form kernel κ, by fixing one of its degrees, i.e., d or c, we can leverage on the considerations made in the previous sections and state that $\kappa^{d,c} \leq_G \kappa^{d+1,c}$ and $\kappa^{d,c} \geq_G \kappa^{d,c+1}$.

However, in more general cases the existence of some kind of order between the degrees is not trivial to demonstrate. A nice observation we can do about DNFs is that, in its extreme case, when c is equal to the maximum number of ones in the vectors, its expressiveness in terms of SR is 1 (its corresponding Gram matrix is indeed an identity matrix). We can also say something similar for the CNF, when d is maximum the SR is 0 (the Gram matrix is a constant matrix). So, in both cases we expect some kind of convergence towards the maximum (resp. minimum) expressiveness.

Empirical results, depicted in Figure 6.7, show that our intuitions are correct. Different curves represent different datasets (see Section 3.2). Moreover, we can also observe two different behaviours between DNF and CNF:

- in the DNF figures we can notice that the expressiveness is monotonically increasing with the degrees and this could suggests that, in general, $\kappa^{d,c}_{\text{(m)DNF}} \leq_G \kappa^{d+1,c+1}_{\text{(m)DNF}}$;

- in the CNF case, the behaviour of the complexity is less predictable. The figures show that with the increasing of the degrees, the expressiveness increases until a certain point, and then starts to decrease monotonically towards 0.

Figure 6.8 summarizes the hierarchical structure of the expressiveness of the (m)DNF-kernel and the (m)CNF-kernel.

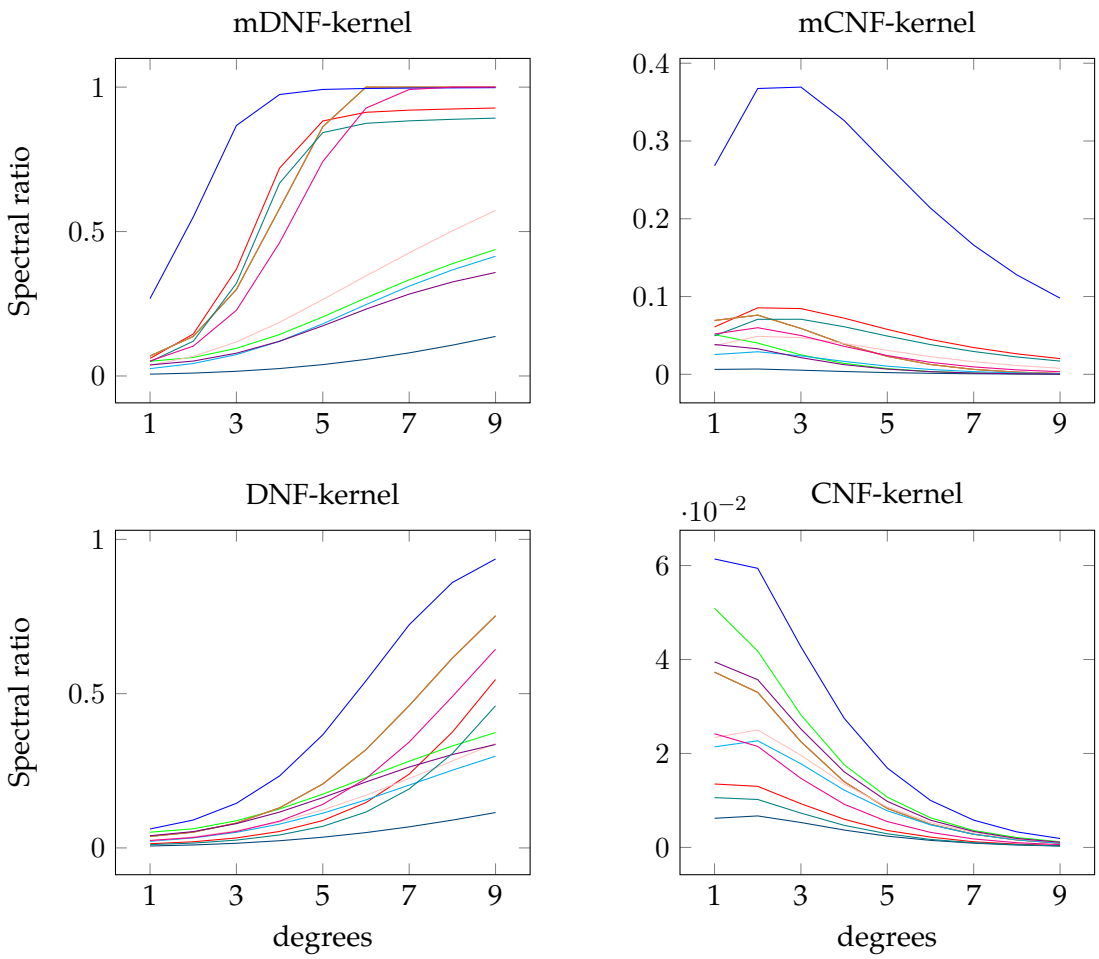


Figure 6.7: Spectral complexity on the benchmark dataset of the normal form kernels varying the degrees d and c, where $d = c$.

6.8 Evaluation

In this section we assess the quality of the Boolean kernels described in this chapter. Firstly, we describe the evaluation setting and then a discussion of the obtained results will follow.

6.8.1 Evaluation protocol

We evaluated the effectiveness of the proposed kernels on several benchmark datasets (Section 3.2) using the standard SVM as classifier. We compared the Boolean kernels with the linear kernel and the prominent RBF kernel (Section 4.2). For each dataset, we

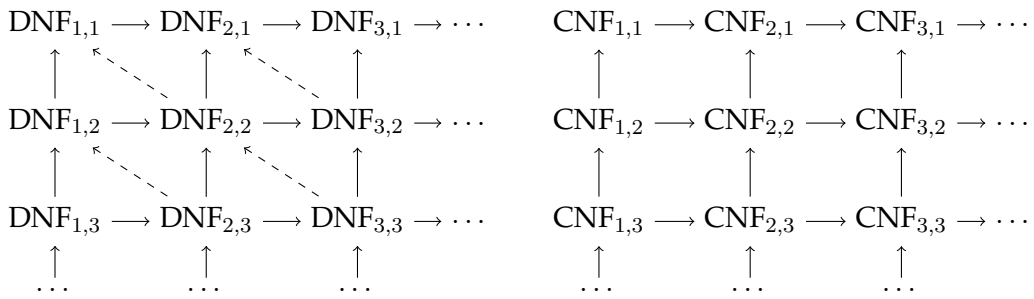


Figure 6.8: Hierarchy of the expressiveness of the (m)DNF-kernel(d,c) (left hand side), and the (m)CNF-kernel(d,c) (right hand side). Solid arrows, e.g., $A \rightarrow B$, indicate that A is more expressive than B, and this relation can be proved by means of Theorem 6.1 and Theorem 6.2. Dashed arrows, $A \dashrightarrow B$, indicate relations that should be valid but that we only have empirical evidence.

performed 20 runs of a 5-fold nested cross validation and the average AUCs with their standard deviations have been recorded. The validated parameters for each kernel are presented in Table 6.2, while the SVM regularization parameter C have been validated in the set of values $\{2^{-5}, 2^{-4} \ldots, 2^{4}\}$.

All these experiments have been made on a MacBook Pro late 2012 with 16GB of RAM and CPU Intel® Core i7 @ 2.70GHz. The methods and the experiments have been implemented in *python* with the machine learning module *Scikit-learn* [PVG+11]. The source code is freely available at the *github* repository `https://github.com/makgyver/pyros`.

Kernel	Parameters
Linear	-
RBF	$\gamma \in \{10^{-4}, \ldots, 10^{3}\}$
(m)C-kernel	$c \in [1, 2, 3, 4, 5]$
(m)D-kernel	$d \in [1, 2, 3, 4, 5]$
NF-kernel	$d \in [1, 2, 3, 4], c \in [1, 2, 3, 4]$

Table 6.2: Validated parameters for the kernels.

The choice of the range of validated degrees for the Boolean kernels has been arbitrarily chosen, however we wanted to use low degrees in order to build "simple" feature spaces.

6.8.2 Experimental results on the artificial datasets

The first set of experiments have been performed on artificial datasets in which the class labels depend on a randomly generated Boolean formula of a specific form. In particular, for every normal form formula and for every pair of degrees (d, c) with $2 \leq d, c \leq 3$, we generated a dataset with 1000 instances of 100 features with a fixed number of active variables, where exactly half of the instances have positive labels (i.e., they satisfy the formula). We created three sets of these datasets to finally get 12 toy datasets per normal form. We omitted formulas with $d = 1$ or $c = 1$ because the tasks were too easy and all the tested kernels performed with the maximum AUC. In these experiments we compared the normal form kernels with the RBF and the linear kernel. The obtained results are reported in Table 6.3. The first evident fact is that the linear kernel is always

Kernel	mDNF	mCNF	DNF	CNF	Rank
Linear	87.87	88.29	91.77	95.06	6.00
RBF	90.42	90.39	93.43	95.68	5.00
mDNF-kernel	90.98	**90.86**	93.92	95.89	2.75
mCNF-kernel	**91.01**	90.81	**93.95**	95.91	2.25
DNF-kernel	91.00	90.84	93.90	**95.92**	**2.00**
CNF-kernel	91.00	90.83	93.88	**95.92**	2.50

Table 6.3: AUC(%) performances on artificial datasets. In **bold**, for each dataset, it is highlighted the best performing kernel. The underlined result indicate the performance of the kernel which corresponds to the formula that generates the dataset. The column "Rank" indicates the average rank of the method w.r.t. the others.

the worst performing one, followed by the RBF kernel which performs, on average, half a point less than the normal form kernels. It is also interesting to notice that all the tested normal form kernels achieved very good performance on all datasets regardless of the formula that generated them. Even though the average rank highlights the DNF-kernel as the best performing kernel, the AUCs of all the proposed kernels are very close to each other. We argue that this can be due to the fact that the expressiveness of the normal form with respect to the target is close between the kernels.

Kernel	dna	house	krkp	prim	promo	soy	spect	spli	t-t-t	mnk2
Linear	98.21 ±0.09	99.38 ±0.17	99.12 ±0.03	73.10 ±1.05	97.41 ±0.96	99.43 ±0.15	84.01 ±1.48	98.48 ±0.05	97.88 ±0.39	48.02 ±3.27
RBF	98.84 ±0.06	**99.39** ±0.20	**99.97** ±0.03	72.82 ±0.94	**97.60** ±0.95	99.61 ±0.14	83.92 ±1.52	99.12 ±0.04	**98.52** ±0.20	99.00 ±0.45
mC	99.03 ±0.04	99.31 ±0.24	99.95 ±0.02	73.04 ±0.97	97.50 ±0.92	99.64 ±0.09	83.96 ±1.43	**99.36** ±0.03•	97.88 ±0.39	**100.0** ±0.00•
mD	98.97 ±0.05	99.32 ±0.27	99.73 ±0.02	72.96 ±0.89	97.50 ±0.94	99.67 ±0.08	84.00 ±1.44	99.19 ±0.04	**97.88** ±0.39	91.09 ±2.27
mDNF	**99.07** ±0.04•	99.30 ±0.25	99.96 ±0.02	72.99 ±0.88	97.57 ±0.96	99.68 ±0.08	83.96 ±1.45	**99.36** ±0.03•	97.88 ±0.39	**100.0** ±0.00•
mCNF	99.04 ±0.04	99.30 ±0.25	99.96 ±0.02	73.07 ±0.83	97.55 ±1.06	**99.69** ±0.08	83.95 ±1.46	**99.36** ±0.03•	97.88 ±0.39	**100.0** ±0.00•
C	99.06 ±0.04•	99.37 ±0.25	99.95 ±0.02	73.05 ±0.88	97.42 ±0.91	99.66 ±0.10	83.96 ±1.41	99.28 ±0.03•	98.38 ±0.26	**100.0** ±0.00•
D	98.96 ±0.05	99.33 ±0.31	99.73 ±0.02	**73.18** ±0.84	97.49 ±0.83	99.66 ±0.07	**84.17** ±1.46	99.21 ±0.04	98.38 ±0.26	87.30 ±3.01
DNF	99.04 ±0.05	99.35 ±0.27	99.96 ±0.02	72.90 ±0.83	97.46 ±0.95	99.67 ±0.08	84.03 ±1.43	99.24 ±0.03	98.38 ±0.26	**100.0** ±0.00•
CNF	99.04 ±0.04	99.35 ±0.27	99.96 ±0.02	72.98 ±0.79	97.46 ±0.97	99.67 ±0.08	83.99 ±1.45	99.25 ±0.03•	98.38 ±0.26	**100.0** ±0.00•

Table 6.4: AUC(%) performances on benchmark datasets with the standard deviation over the 20 runs. In **bold**, for each dataset, it is highlighted the best performing kernel. The • near the standard deviation indicates a performance that is significantly better than RBF (more than two standard deviations).

6.8.3 Experimental results on the benchmark datasets

With the following experiments we want to assess the quality of our proposed kernel functions on binary classification tasks on benchmark datasets. The performance achieved by the different kernels are reported in Table 6.4. Note that the datasets `monks-1` and `monks-3` are not present in the table for space reasons. However, all the results are also depicted in Figure 6.9.

A first observation about the results is that the normal form kernels achieved usually very good performance w.r.t. the other kernels. It is worth noticing that the normal form kernels are always better than their correspondent "base kernels" (disjunctives and conjunctives) and this is due to the fact that the normal forms are a generalizations of their base kernels. Moreover, this behaviour is also a confirmation that the validation had

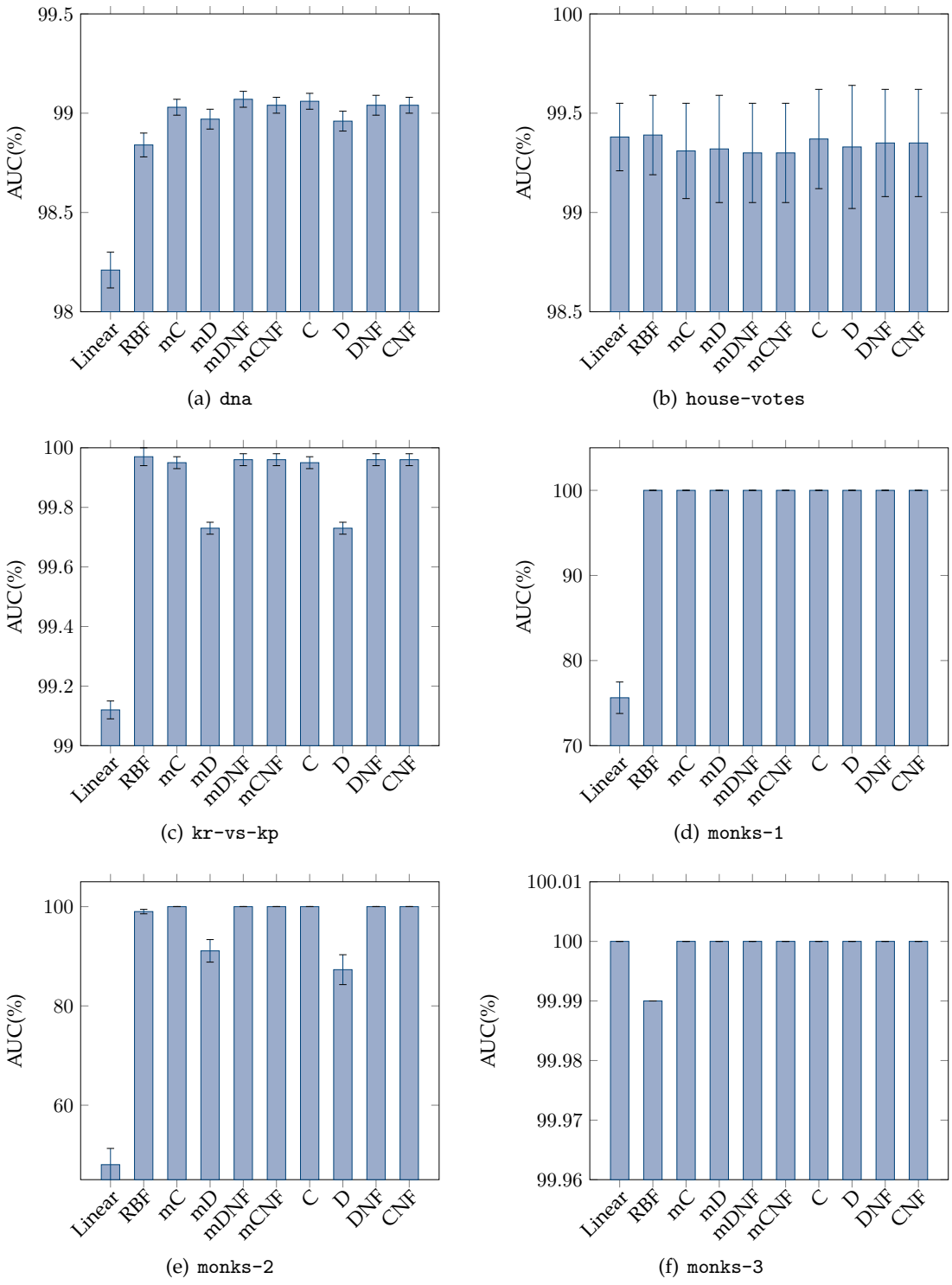
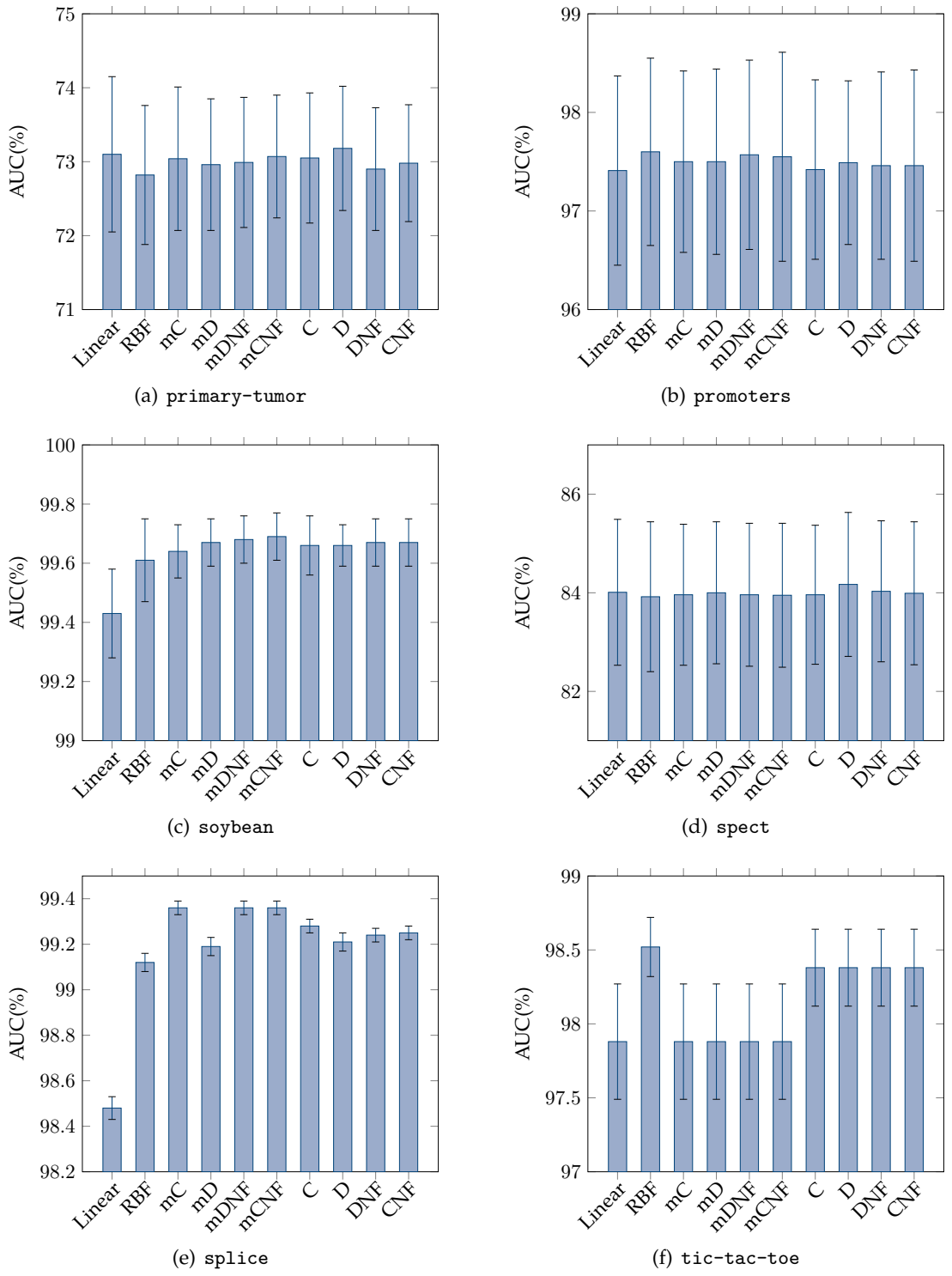


Figure 6.9: AUC results on many benchmark datasets: the proposed kernels are compared to the linear kernel and the RBF kernel.



(a) `primary-tumor` (b) `promoters`

(c) `soybean` (d) `spect`

(e) `splice` (f) `tic-tac-toe`

Figure 6.9: AUC results on many benchmark datasets: the proposed kernels are compared to the linear kernel and the RBF kernel.

worked properly. Unsurprisingly, the linear kernel is in general (7 out of 12) the worst performing one because it is unable to adequately represent the non linear relations in the data. Conversely, as expected, the RBF kernel achieves very good AUC scores even though it is not as good as our Boolean kernels in some of the datasets, specifically `dna`, `monks-2`, `monks-3` and `splice` in which the RBF-kernel performance is significantly worse than the one of our proposals.

A peculiarity of the `monks` datasets is that their labelling can be explained by DNF rules. The obtained results on these datasets show that most of the proposed Boolean kernels achieve an AUC of 100% while RBF, despite it has a good performance, is not perfect. This is due to the fact that these Boolean kernels contain the target formula in the feature space, or at least they contain a set of related formulas that combined together are able to mimic the target one.

On average, the worst performing Boolean kernels are the disjunctives (both monotone and non-monotone) and this can be easily explained by their poor expressiveness (see Section 4.6).

These results show that our Boolean kernels can achieve state-of-the-art performance on categorical datasets, with the convenience of creating embedding spaces that are easy to interpret, and this gives the opportunity to apply rule extraction approaches directly on the feature space.

7

Interpreting SVM

> What is the most resilient parasite? Bacteria? A virus? An intestinal worm? An idea. Resilient... highly contagious. Once an idea has taken hold of the brain it's almost impossible to eradicate. An idea that is fully formed - fully understood - that sticks; (taps forehead) right in there somewhere.
>
> *Inception [talking about extraction]*
> Dominic "Dom" Cobb

In short

In this chapter we propose a simple way for extracting rules from the solution of a kernel method using Boolean kernels. First of all, we provide the theoretical foundations upon which our method is based. Then, we present a proof of concept method and we show its applicability on some datasets. The proposed method is based on the general purpose framework concerning genetic algorithms. What we want to underline here is that interpreting kernel machines based on Boolean kernels is feasible.

7.1 Features relevance in the feature space

In the previous sections of this chapter we have given a series of kernels for which we can easily interpret their features in the embedding space. Now, our goal is to find a way to extract from such feature spaces those features that are the most relevant in the decision of a kernel machine. This task can be seen as a feature relevance/feature selection problem.

Several definitions of relevance have been suggested in the literature, for example, Almuallim et al. [AD91] give the following definition of relevance, under the assumptions that all features and the label are Boolean and that there is no noise.

Definition 7.1 : Relevant Feature [AD91]

A feature X_i is said to be relevant to a concept C if X_i appears in every Boolean formula that represents C and irrelevant otherwise.

Definition 7.1 correlates the relevance of a feature to the target concept, while there are other definition that relate the relevance to the behaviour of a particular hypothesis h.

Definition 7.2 : Relevant feature for h [NPnBT07]

A feature X_i is relevant to the hypothesis h if

$$\mathbb{P}(h(X_i, F_i) \neq h(X_i', F_i)) > 0$$

where X_i and X_i' are independent samples from the marginal distribution of the feature X_i, and $F_i \equiv \{X_1, X_2, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n\}$ is the set of all features but X_i.

In other words, Definition 7.2 states that a feature is relevant to an hypothesis h if re-sampling such feature according to its marginal distribution influences the behaviour of the classifier with a non-zero probability. Such definition has also been extended to hypothesis class, in which a feature is relevant for the hypothesis class $\mathcal{H}$ if it is relevant for all hypothesis $h \in \mathcal{H}$ in the sense of Definition 7.2 [GPH17].

However, the underlying data distribution is unknown and hence it is impossible to say exactly whether a feature is relevant for an hypothesis. For this reason a possible approach is to use some heuristics to estimate the relevance. In [GPH17], Göpfert et al. gave a very simple heuristic for the feature relevance for the hypothesis class of linear classifier. In particular, given a linear classifier, i.e, an hyperplane, defined by the pair ($\mathbf{w}$, b), the relevance of a feature f can be calculated by the absolute value of its corresponding weights in $\mathbf{w}$, that is $\mathbf{w}_f$.

By relying on a similar heuristic, we define the relevance of a feature in the feature space as in the following. Let us consider an hypothesis from the solution of an SVM using a Boolean kernel as kernel function over a dataset $\mathcal{D} \equiv \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$. We know from the Representer Theorem 4.3.2 that the hyperplane of a SVM can be written as

$$\mathbf{w} = \sum_{i \in \mathcal{S}} y_i \boldsymbol{\alpha}_i \phi(\mathbf{x}_i)$$

with $\mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^-$ is the set of support vector indexes where $\mathcal{S}^+$ ($\mathcal{S}^-$) the set of positive (respectively, negative) support vector indexes.

Let us now consider for simplicity examples of fixed norm in feature space, i.e., $\|\phi(\mathbf{x})\| = q$, and let $\delta_f(\mathbf{x}) \in \{0,1\}$ be the value of the feature f (i.e., a logic formula in the case of Boolean kernels) in the feature space of $\mathbf{x}$. Then the associated weight for a feature f can be calculated by:

$$\begin{aligned} w_f &= \frac{1}{q}\sum_{i\in\mathcal{S}} y_i\boldsymbol{\alpha}_i\delta_f(\mathbf{x}_i) \\ &= \frac{1}{q}\sum_{i\in\mathcal{S}^+}\boldsymbol{\alpha}_i\delta_f(\mathbf{x}_i) - \frac{1}{q}\sum_{i\in\mathcal{S}^-}\boldsymbol{\alpha}_i\delta_f(\mathbf{x}_i), \end{aligned} \tag{7.1}$$

where $\boldsymbol{\alpha}_i$ are the contributions of the support vectors to the solution of the SVM.

Our aim is to find the formula f such to maximize the value w_f. Since all $\boldsymbol{\alpha}_i$ are positives and $\sum_{i=1}^{l} y_i\boldsymbol{\alpha}_i = 0$, then this problem can be reduced to the one of finding the formula such that $w_f = q^{-1}\sum_{i\in\mathcal{S}^+}\boldsymbol{\alpha}_i$. It is easy to show that if (i) the set is separable, (ii) the target concept is given by a target formula g, and (iii) the feature space contains g, then

$$w_g = \operatorname*{argmax}_f w_f = \frac{1}{q}\sum_{i\in\mathcal{S}^+}\boldsymbol{\alpha}_i. \tag{7.2}$$

This is due to the fact that the target function generates the labels, then $\delta_f(\mathbf{x}_i) = 1 \Leftrightarrow y_i = +1$. Moreover, any other formula f for which it holds has $w_f = w_g$.

In the case of non-separability, maximizing the value of w_f is still a good heuristic since it is a way to minimize the loss with respect to the decision function.

7.2 Interpreting BK-SVM

This section describes how Boolean kernels can be used in order to interpret a kernel machine solution, such as anSVM. First of all we present a new Boolean kernel computed as combination of mC-kernels and mD-kernels. Then, an algorithm for interpreting a SVM based on Boolean kernels (BK-SVM) will be presented.

7.2.1 The feature space of monotone DNFs

As described in Chapter 6, by composing the mC-kernel and the mD-kernel it is possible to define the mDNF-kernel in which the feature space is composed by monotone DNF formulas over the input variables. A shortcoming of the mDNF-kernel defined in such way is that the mDNF formulas have a fixed form, that is, they are composed by disjunctions of d conjunctive clauses made of c literals. In order to overcome this

limitation, instead of directly composing the feature maps of the mC-kernel and the mD-kernel, we first create a feature space composed by all conjunctions up to a certain arity C, by concatenating the feature spaces of κ^c_{mC} for $c \in [1, C]$, that is

$$\phi^C_\Sigma(\mathbf{x}) = (\phi^1_{\text{mC}}(\mathbf{x}), \phi^2_{\text{mC}}(\mathbf{x}), \ldots, \phi^C_{\text{mC}}(\mathbf{x})).$$

The corresponding kernel can be implicitly computed [STC04] by

$$\kappa^C_\Sigma(\mathbf{x}, \mathbf{z}) = \sum_{c=1}^{C} \kappa^c_{\text{mC}}(\mathbf{x}, \mathbf{z}).$$

Now, by composing ϕ^C_Σ with ϕ^d_{mD} (for some d) we obtain a feature space constituted of all possible mDNFs made of d conjunctive clauses of at most C literals. This kernel can be calculated by replacing $\langle \mathbf{x}, \mathbf{z} \rangle$ (i.e., the linear kernel) with $\kappa^C_\Sigma(\mathbf{x}, \mathbf{z})$ and n with $\sum_{c=1}^{C} \binom{n}{c}$ in Eq. (6.7). Finally, by summing up all these kernels with $d \in [1, D]$ we obtain a kernel with a feature space composed of all possible mDNF formulas with at most D conjunctive clauses of at most C literals. Formally,

$$\kappa^{D,C}_*(\mathbf{x}, \mathbf{z}) = \sum_{d=1}^{C} \kappa^d_{\text{mD}}(\phi^C_\Sigma(\mathbf{x}), \phi^C_\Sigma(\mathbf{z})).$$

7.2.2 Rule extraction via Genetic Algorithm

Finding the best feature is not an easy task because of the huge dimensionality of the feature space and hence, in general, an exhaustive search is not feasible. For this reason, we adopted a genetic algorithm (GA) based optimization. The design choices for the GA are described in the following:

population it is formed by 500 randomly initialized individuals, i.e., mDNF formulas with at most D conjunctions made of at most C literals;

fitness given a formula f, its fitness is equals to the weight w_f as in Eq. (7.2);

crossover given two mDNF formulas f and g, the crossover operator creates a new individual by randomly selecting a subset of the conjunctive clauses from the union of f and g while keeping the number of clauses $\leq D$.

mutation given a mDNF formula, the mutation operator randomly performs one out of the following three actions: (i) removing one of the conjunctive clauses (when applicable); (ii) adding a new random conjunctive clause; (iii) permuting a literal in one of the conjunctions with another literal picked from the ones that are not currently included in it;

selection we adopted the *elitist selection* (20%) strategy to guarantee that the solution quality will not decrease.

7.3 Experiments

7.3.1 Experimental settings

The experiments have been performed on 10 binary datasets in which the number of ones is the same for every instance. This is not a limitation since, given a dataset with categorical features, each instance can be converted into a fixed norm binary vector by means of the one-hot encoding [HH13]. The artificial datasets (indicated by the prefix `art-`) have been created in such a way that the positive class can be described by a mDNF formula over the input variables. All the experiments have been implemented in python 2.7 using the modules Scikit-Learn, MKLpy and pyros available in the PyPi repository.

The details of the datasets are summarized in Table 7.1. We evaluated the proposed

Dataset	#Inst.	#Ft.	Rule
`tic-tac-toe`	958	27	mDNF, $d = 8, c = 3$
`monks-1`	432	17	$(x_0 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee x_{11}$
`monks-3`	432	17	mDNF, $d = 7, c = 2$
`art-d2-c4`	1000	30	$(x_{11} \wedge x_9 \wedge x_1 \wedge x_{14}) \vee (x_{27} \wedge x_{17})$
`art-d3-c3`	1000	30	$(x_3 \wedge x_{28}) \vee x_{27} \vee (x_{14} \wedge x_7 \wedge x_{26})$
`art-d4-c2`	1000	30	$x_3 \vee (x_0 \wedge x_7) \vee (x_5 \wedge x_9) \vee x_8$
`art-d4-c3`	1000	30	$(x_{25} \wedge x_{21}) \vee (x_{15} \wedge x_5 \wedge x_{19}) \vee (x_0 \wedge x_{26}) \vee (x_8 \wedge x_{21} \wedge x_{20})$
`art-d5-c4`	1000	30	mDNF, $d = 5, c \leq 4$
`art-d5-c5`	1000	30	mDNF, $d = 5, c \leq 5$

Table 7.1: Information of the datasets: number of instances, number of binary features and the rule which describes the positive class.

algorithm in terms of the most used metrics for evaluating explanation rules [BB10], namely, *comprehensibility*, *accuracy* and *fidelity*. Comprehensibility is the extent to which the extracted representations are humanly comprehensible. In our case we can assume high comprehensibility because the retrieved rules are simple (and short) logical propositions over the input binary variables. The accuracy of a classification function (or rule)

f over the test set $\mathcal{T}_{\text{ts}}$ is equal to

$$\text{accuracy}(f, \mathcal{T}_{\text{ts}}) = \frac{|\{(\mathbf{x}, y) \in \mathcal{T}_{\text{ts}} \mid [\![f(\mathbf{x}) \iff y = +1]\!]\}}{|\mathcal{T}_{\text{ts}}|}.$$

The *fidelity* over the test set $\mathcal{T}_{\text{ts}}$ of a rule f w.r.t. a decision function h learnt by a learning algorithm is computed by

$$\text{fidelity}(f, h, \mathcal{T}_{\text{ts}}) = \frac{|\{(\mathbf{x}, y) \in \mathcal{T}_{\text{ts}} \mid [\![f(\mathbf{x}) \iff h(\mathbf{x}) = +1]\!]\}|}{|\mathcal{T}_{\text{ts}}|}.$$

For each dataset the experiments have been repeated 5 times by using different 70%-30% training-test splits. In each experiment an hard-SVM with the kernel $\kappa_*^{5,10}$ has been trained over the training set and then the most relevant formula has been extracted using the GA (described in Section 7.2.2) with $C = 5$, $D = 10$, the mutation probability set to 0.6 and the maximum number of generations set to 10^3.

7.3.2 Results

The achieved results are summarized in Table 7.2. As evident from the table, in every dataset the best rule extracted by the GA is indeed the one which (almost always) explains the label and the decision of the SVM (the fidelity is very high). Moreover, despite the huge search space (on average 10^{45} formulas), the number of generations required to find the best rule is very low.

To highlight how the weights learned by the SVM are indeed useful to guide the research of the GA (through the fitness), we also tried to retrieve the best formula by using the same GA with $\alpha_i = 1/L, \forall\, i \in [1, L]$. In this case the fitness corresponds to the training accuracy. Figures 7.1 and 7.2 show the comparison between the GA w/ and w/o SVM. From the figures, it is evident that using the GA guided by the SVM ensures that a better rule will be found with fewer generations. It is also worth to mention that computing the fitness over all the training set is significantly less efficient than calculating it for the support vectors only.

	SVM		Best Rule		Fidelity		
Dataset	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**GA #Gen.**
tic-tac-toe	100.00 ±0.00	98.33 ±0.87	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	98.33 ±0.87	358.00 ±156.81
monks-1	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	9.20 ±3.37
monks-3	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	230.40 ±385.79
art-d2-c4	100.00 ±0.00	98.87 ±0.50	99.89 ±0.23	99.40 ±0.80	99.89 ±0.23	99.07 ±0.68	10.60 ±3.55
art-d3-c3	100.00 ±0.00	97.13 ±1.13	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	97.13 ±1.13	14.60 ±7.34
art-d4-c2	100.00 ±0.00	97.87 ±0.75	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	97.87 ±0.75	15.0 ±2.28
art-d4-c3	100.00 ±0.00	95.07 ±1.34	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	95.07 ±1.34	35.20 ±19.36
art-d5-c4	100.00 ±0.00	96.00 ±0.67	99.71 ±0.57	99.20 ±1.60	99.71 ±0.57	96.00 ±0.67	340.40 ±338.10
art-d5-c5	100.00 ±0.00	94.27 ±0.85	99.97 ±0.06	99.40 ±0.33	99.97 ±0.06	94.20 ±0.85	61.20 ±17.68

Table 7.2: Experimental results averaged over 5 runs: for each dataset the accuracy (%) in both training and test is reported for SVM and for the extracted rule. It is also reported the fidelity of the rule w.r.t the SVM as well as the average number of generations required to the GA to find the best rule.

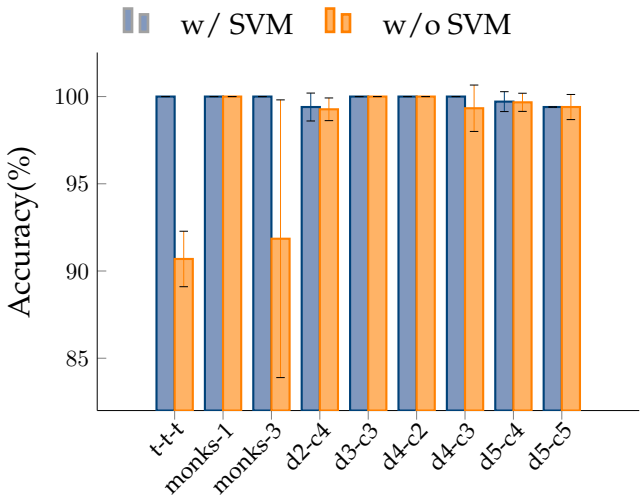


Figure 7.1: Comparison between the GA guided by the SVM (w/) and w/o the SVM. The plot shows the average accuracy on the test set.

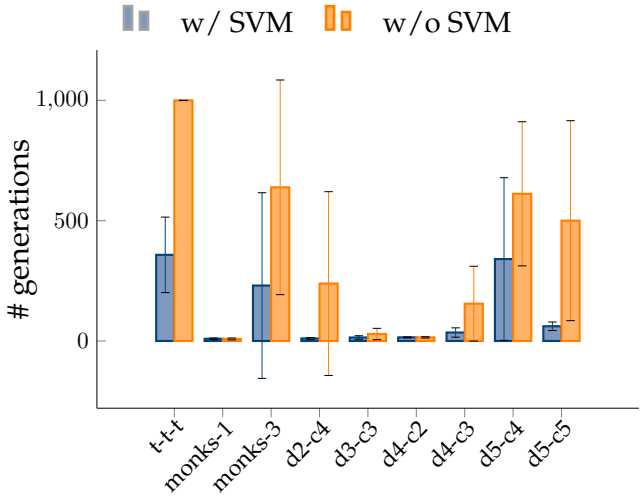


Figure 7.2: Comparison between the GA guided by the SVM (w/) and w/o the SVM. The plot shows the average number of generations required by the GA to find the best rule.

8

Propositional kernels

Logic takes care of itself; all we have to do is to look and see how it does it.

Ludwig Wittgenstein

In short

This chapter presents a new kernel family related to the Boolean kernels, but that overcome their expressiveness limitation imposed in the design in order to get easy-to-interpret feature spaces. This novel framework, dubbed propositional kernel framework, allows to create feature spaces with any possible structure of logical formulas. It is clear that this flexibility makes their interpretation harder, however they can express logical concept that Boolean kernels cannot. We also provide an algorithm for generating propositional kernels, and by using it we show examples of application on several binary classification problems.

8.1 Limitations of the Boolean kernels

In Chapter 6 we have proposed a set of Boolean kernels designed to produce interpretable feature spaces composed of logical formulas over the input variables. However, this nice characteristic comes with a cost: the set of possible logical formulas that can be formed is limited. In particular, there are two aspects which limit the logical expressiveness of the Boolean kernels:

(i) they do not consider clauses with the same variable repeated more than once. Even though such feature is, from a logical point of view, appropriate, it cause many issues when we have to formally define the kernel functions, since we need

to introduce the binomial coefficient which makes harder both the reasoning and the computation;

(ii) they are "symmetric": each Boolean concept, described by a feature in the embedding space, is symmetric in their clauses. For example, an mDNF-kernel(3,2) creates mDNF formulas that are disjunctions of 3 conjunctive clauses of 2 variables. So, every single conjunctive clause is in some sense symmetric to the others (each of them have exactly 2 variables). It is not possible to form an mDNF of the form $(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_4)$ where different conjunctive clauses have different arity.

For these reasons, in this chapter we propose a framework to produce kernels with a corresponding feature space that can potentially express any fixed logical proposition over the input variables.

In order to accomplish our goal, we need to overcome the limitations of the Boolean kernels, and we have also to provide a way to construct any possible logical formulas.

Overcoming the first limitation of the Boolean kernels, i.e., no repeated variables in the formulas, is very simple: it is sufficient to include any possible combination, even those ones that are logically a contradiction or a tautology.

Regarding the symmetry, some considerations need to be done. Let us assume we want to create a kernel function such that its feature space is composed of mDNF of the form $f(a, b, c) = (a \wedge b) \vee c$, using the Boolean kernels. Recalling the definition in Section 6.6.1, the embedding map of an mDNF-kernel is defined as the composition of the embedding maps of the mD-kernel and the mC-kernel as:

$$\widetilde{\phi}_{\text{mDNF}} : \mathbf{x} \mapsto \widetilde{\phi}_{\text{mD}}(\widetilde{\phi}_{\text{mC}}(\mathbf{x})),$$

where we have omitted the degrees and put a ~ over the functions to emphasize that we do not want to be linked to specific degrees. By this definition, there is no way to get a feature space with only formulas like f since we would need conjunctive clauses with different degrees, which is not possible. Now, let say we redefine $\widetilde{\phi}_{\text{mC}}$, in such a way that it can contain both conjunctions of degree 1 and degree 2, for example, by summing an mC-kernel of degree 1 and an mC-kernel of degree 2. The resulting mapping $\widetilde{\phi}_{\text{mDNF}}$ would not create an embedding space with only f-like formulas anyway, because it would also contain formulas like $(a) \vee b$. Unfortunately, we cannot overcome this last issue using Boolean kernels in the way they are defined.

The main problem originates from the basic idea behind Boolean kernels, that is creating logical formulas "reusing" the same set of inputs in each literal. For example, let us consider the disjunction $(a \vee b)$ and let us try to construct the feature space of such formulas using the Boolean kernels. Given an input binary vector $\mathbf{x}$, the first literal of the disjunction, i.e., a, can be any variables of $\mathbf{x}$ and the same can be said for b. It is clear

that we cannot break this symmetry since we are taking both literals from the same pool of variables.

8.2 Propositional kernels formulation

On the basis of the observations made in the previous section, we now give the intuition behind the construction of the propositional kernels.

Let us take into consideration logical formulas of the form $f_{\otimes}(a, b) = a \otimes b$ where $\otimes$ is some Boolean operation over the variables $a, b \in \{0, 1\}$. In order to construct formulas in such a way that a is taken from a set of variables and b from (possibly) another set, we need to consider two different input Boolean domains, that we will call A and B, respectively. These domains are intended as sets of other Boolean formulas over the input variables. Now, given an input vector $\mathbf{x} \in \{0, 1\}^n$, we map $\mathbf{x}$ in both the domain A and B, i.e., $\phi_{\text{A}}(\mathbf{x})$ and $\phi_{\text{B}}(\mathbf{x})$, and then we perform the Boolean function $f_{\otimes}$ by taking one variable from $\phi_{\text{A}}(\mathbf{x})$ and one from $\phi_{\text{B}}(\mathbf{x})$. Figure 8.1 graphically shows the above described procedure.

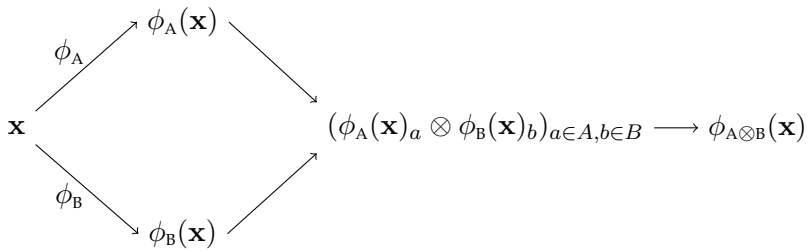


Figure 8.1: A graphical depiction of the idea behind the propositional kernels for breaking the symmetry of the Boolean kernels: the input vector is mapped onto two, potentially different, spaces, and then the final feature space is composed of all possible pairs of features one taken from the space of ϕ_A and one taken from the space of ϕ_B. Finally the logical interpretation of such final features depend on the operation we want to perform.

Formally, we can define a generic propositional kernel embedding function for the logical operator $\otimes$ over the domains A and B as:

$$\phi_{\text{A}\otimes\text{B}}(\mathbf{x}) : \mathbf{x} \mapsto (\phi_{\text{A}}(\mathbf{x})_a \otimes \phi_{\text{B}}(\mathbf{x})_b)_{a\in A, b\in B}, \tag{8.1}$$

and consequently the corresponding kernel function $\kappa_{\text{A}\otimes\text{B}}(\mathbf{x}, \mathbf{z})$ is defined by

$$\kappa_{\text{A}\otimes\text{B}}(\mathbf{x}, \mathbf{z}) = \langle \phi_{\text{A}\otimes\text{B}}(\mathbf{x}), \phi_{\text{A}\otimes\text{B}}(\mathbf{z}) \rangle = \sum_{(a,b)\in A\times B} (\phi_{\text{A}}(\mathbf{x})_a \otimes \phi_{\text{B}}(\mathbf{x})_b)(\phi_{\text{A}}(\mathbf{z})_a \otimes \phi_{\text{B}}(\mathbf{z})_b). \tag{8.2}$$

The kernel $\kappa_{\text{A}\otimes\text{B}}(\mathbf{x}, \mathbf{z})$ counts how many logical formulas of the form $a \otimes b$, with a taken from the feature space of ϕ_{A} and b taken from the feature space of ϕ_{B}, are true in both $\mathbf{x}$ and $\mathbf{z}$. To check whether this formulation is ideally able to build kernels for a generic Boolean formulas, let us reconsider the example of the previous section, i.e., $f(a, b, c) = (a \wedge b) \vee c$. If we assume that the domain A contains all the formulas of the form $(a \wedge b)$, while the domain B contains single literals (actually it corresponds to the input space), then using Equation (8.2) we can define a kernel for f by simply posing $\otimes \equiv \vee$. It is easy to see that, by using similar considerations as in the example above, we can derive any kernel. However, we need to design a method to compute it without expliciting any of the involved spaces (with the exception of the input space).

8.3 Construction of a propositional kernel

Since we want to be as much general as possible, we need to define a constructive method for generating and computing a propositional kernel, rather than a specific formulations for any possible different formula.

For doing this, we leverage on a well known result which states that Boolean formulas can be defined as strings generated by a context-free grammar.

Definition 8.1 : Grammar for propositional logic [BA12]

Formula in propositional logic are derived from the context-free grammar, $G_{\mathcal{P}}$, whose terminals are:

- a set of symbols $\mathcal{P}$ called *atomic propositions*;
- a set $\mathcal{B}$ of Boolean operators.

The context-free grammar $G_{\mathcal{P}}$ is defined by the following productions:

$$F ::= p, \quad p \in \mathcal{P}$$
$$F ::= \neg F$$
$$F ::= F \otimes F, \quad \otimes \in \mathcal{B}$$

A formula is a word that can be derived from the nonterminal F.

Starting from the grammar $G_{\mathcal{P}}$, we can define the propositional kernel framework by providing a kernel for each production, and then, with simple combinations, we will be able to build any propositional kernel by following the rules of the grammar.

The first production, i.e., $F ::= p$, is trivial since is the monotone literal kernel (κ_{mL})

presented in Chapter 6. Similarly, the second production, i.e., $F ::= \neg F$, which represents the negation, corresponds to the negation kernel (κ_{N}) which has been also presented in Chapter 6.

The third and last production, i.e., $F ::= F \otimes F$, is the trickiest one: it represents a generic binary operation between two logical formulas.

In order to be general with respect to the operation $F \otimes F$, we need a way for distinguishing the two operands and, moreover, we cannot make any assumption about what the operator represents. For these reasons, we will refer to the first and the second operand with A and B, respectively. Regarding the operation $\otimes$, we consider a generic truth table as define in Table 8.1, where $y_{\otimes}(a, b)$ is the truth value given all possible combinations of $a, b \in \{0, 1\}$.

a	b	$y_{\otimes}(a, b)$
0	0	$0 \otimes 0$
0	1	$0 \otimes 1$
1	0	$1 \otimes 0$
1	1	$1 \otimes 1$

Table 8.1: Generic truth table for the $\otimes$ operation.

It is easy to see that the kernel we want to define for the operation $\otimes$ has exactly the form of the kernel $\kappa_{\text{A}\otimes\text{B}}(\mathbf{x}, \mathbf{z})$ described in the previous section: each operand is taken from (potentially) different spaces, i.e., different formulas, and the kernel counts how many of these logical operations are true in both the input vectors.

Since we have to count the common true formulas, we need to take into account the configurations of a and b that generate a true value for $y_{\otimes}(a, b)$, and given those true configurations we have to consider all the true joint configurations between the inputs. In other words, a formula can be true for $\mathbf{x}$ from a certain configuration while it can be also true for $\mathbf{z}$ from another configuration. For example, let the formula be a disjunction of arity 2 ($a \vee b$), then given a feature of the embedding space this can be true for $\mathbf{x}$ because its a-part is true, and vice versa for $\mathbf{z}$. To clarify this last concept, please consider the Table 8.2.

It is evident that the value of the kernel is the sum over all the possible joint configurations of the common true target values between $\mathbf{x}$ and $\mathbf{z}$. In order to compute that, for each row of the Table 8.2 we calculate the true formulas in $\mathbf{x}$ and $\mathbf{z}$ for the configuration corresponding to the row. For example, in the first row of the table we have to count all the common true formulas such that the features in A and in B are false in both $\mathbf{x}$ and $\mathbf{z}$, and this can be computed by:

$$(\mathbf{1}_{|A|} - \phi_{\text{A}}(\mathbf{x}))^{\intercal}(\mathbf{1}_{|A|} - \phi_{\text{A}}(\mathbf{z}))(\mathbf{1}_{|B|} - \phi_{\text{B}}(\mathbf{x}))^{\intercal}(\mathbf{1}_{|B|} - \phi_{\text{B}}(\mathbf{z})),$$

$a_{\mathbf{x}}$	$b_{\mathbf{x}}$	$a_{\mathbf{z}}$	$b_{\mathbf{z}}$	$y_{\otimes}(a_{\mathbf{x}}, b_{\mathbf{x}})y_{\otimes}(a_{\mathbf{z}}, b_{\mathbf{z}})$
0	0	0	0	$(0 \otimes 0)(0 \otimes 0)$
0	1	0	0	$(0 \otimes 1)(0 \otimes 0)$
1	0	0	0	$(1 \otimes 0)(0 \otimes 0)$
1	1	0	0	$(1 \otimes 1)(0 \otimes 0)$
0	0	0	1	$(0 \otimes 0)(0 \otimes 1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	1	1	$(1 \otimes 1)(1 \otimes 1)$

Table 8.2: Joint truth table between $\mathbf{x}$ and $\mathbf{z}$ for the operation $\otimes$.

which is actually the product of the negation kernels in the domain A and B, that is:

$$\kappa_{\text{NEG}}(\phi_{\text{A}}(\mathbf{x}), \phi_{\text{A}}(\mathbf{z}))\kappa_{\text{NEG}}(\phi_{\text{B}}(\mathbf{x}), \phi_{\text{B}}(\mathbf{z})).$$

Such computation can be generalized over all the possible 16 configurations, i.e., the rows of the joint truth table, by the following formula

$$\kappa_{\text{A}\otimes\text{B}}(\mathbf{x}, \mathbf{z}) = \sum_{(a_{\mathbf{x}}, b_{\mathbf{x}})\in\mathbb{B}} \sum_{(a_{\mathbf{z}}, b_{\mathbf{z}})\in\mathbb{B}} y_{\otimes}(a_{\mathbf{x}}, b_{\mathbf{x}})y_{\otimes}(a_{\mathbf{z}}, b_{\mathbf{z}})\Psi_{\text{A}}(\mathbf{x}, a_{\mathbf{x}})^{\intercal}\Psi_{\text{A}}(\mathbf{z}, a_{\mathbf{z}})\Psi_{\text{B}}(\mathbf{x}, a_{\mathbf{x}})^{\intercal}\Psi_{\text{B}}(\mathbf{z}, b_{\mathbf{z}}) \tag{8.3}$$

where $\mathbb{B} \equiv \{(a, b) \mid a \in \{0, 1\}, b \in \{0, 1\}\}$ and $\Psi_{\text{A}} : \{0, 1\}^n \times \{0, 1\} \to \{0, 1\}^{|A|}$ is defined as

$$\Psi_{\text{A}}(\mathbf{x}, a_{\mathbf{x}}) = (1 - a_{\mathbf{x}})\mathbf{1}_{|A|} + (2a_{\mathbf{x}} - 1)\phi_{\text{A}}(\mathbf{x}) = \begin{cases} \phi_{\text{A}}(\mathbf{x}) & \text{if } a_{\mathbf{x}} = 1 \\ \mathbf{1}_{|A|} - \phi_{\text{A}}(\mathbf{x}) & \text{if } a_{\mathbf{x}} = 0 \end{cases},$$

and the definition is analogous for Ψ_{B}.

In its worst case, that is when the joint truth table has 1 in every configuration, the formula has 16 non-zero terms. However, we have to underline that actually only a small set of operations need the computation of the corresponding kernel via Equation (8.3) since we can use logic equivalences and apply them with the propositional kernels. The only exceptions where the logic equivalences do not hold for the propositional kernels is when there are constraints in the variables. For example, in logic we can express the *xor* operation by means of *and*, *or* and *not*, i.e., $a \oplus b \leftrightarrow (a \wedge \neg b) \vee (\neg a \wedge b)$, but this cannot be done with kernels since we have no way to fix the relations between the first conjunctive clause and the second conjunctive clause. In all the other cases, logic equivalences hold, e.g., the De Morgan's laws and the double negation rule, and this allows to compute, for example, the implication kernel in terms of the disjunctive propositional kernel and the negation kernel.

In the following we provide a couple of instantiation examples of Equation (8.3) for computing the propositional kernels.

Example of conjunction

The truth table of the conjunction has only one true output, that is $y_{\wedge}(1,1)$. Hence, there exists a unique term in the summation of $\kappa_{\mathrm{A}\wedge\mathrm{B}}$ s.t. $y_{\wedge}(a_{\mathbf{x}}, b_{\mathbf{x}})y_{\wedge}(a_{\mathbf{z}}, b_{\mathbf{z}}) = 1$, that is when $a_{\mathbf{x}} = b_{\mathbf{x}} = a_{\mathbf{z}} = b_{\mathbf{z}} = 1$. This leads to the following formulation

$$\begin{aligned}\kappa_{\mathrm{A}\wedge\mathrm{B}}(\mathbf{x},\mathbf{z}) &= \Psi_{\mathrm{A}}(\mathbf{x},1)^{\intercal}\Psi_{\mathrm{A}}(\mathbf{z},1)\Psi_{\mathrm{B}}(\mathbf{x},1)^{\intercal}\Psi_{\mathrm{B}}(\mathbf{z},1)\\ &= \phi_{\mathrm{A}}(\mathbf{x})^{\intercal}\phi_{\mathrm{A}}(\mathbf{z}) \cdot \phi_{\mathrm{B}}(\mathbf{x})^{\intercal}\phi_{\mathrm{B}}(\mathbf{z})\\ &= \kappa_{\mathrm{A}}(\mathbf{x},\mathbf{z})\kappa_{\mathrm{B}}(\mathbf{x},\mathbf{z}),\end{aligned}$$

which is actually the number of possible conjunctions $a \wedge b$ that are satisfied in both $\mathbf{x}$ and $\mathbf{z}$ s.t. $a \in A, b \in B$. This can be defined as the product between the number of common true formulas in A and the number of common true formulas in B, that is the product of the kernels κ_{A} and κ_{B}.

Example of exclusive disjunction

The truth table of the exclusive disjunction has two true outputs, that is when a and b have different truth values. So in this case, the joint truth table have four non-zero terms:

$$\begin{aligned}\kappa_{\mathrm{A}\oplus\mathrm{B}}(\mathbf{x},\mathbf{z}) =& \Psi_{\mathrm{A}}(\mathbf{x},1)^{\intercal}\Psi_{\mathrm{A}}(\mathbf{z},1)\Psi_{\mathrm{B}}(\mathbf{x},0)^{\intercal}\Psi_{\mathrm{B}}(\mathbf{z},0)+\\ & \Psi_{\mathrm{A}}(\mathbf{x},1)^{\intercal}\Psi_{\mathrm{A}}(\mathbf{z},0)\Psi_{\mathrm{B}}(\mathbf{x},0)^{\intercal}\Psi_{\mathrm{B}}(\mathbf{z},1)+\\ & \Psi_{\mathrm{A}}(\mathbf{x},0)^{\intercal}\Psi_{\mathrm{A}}(\mathbf{z},1)\Psi_{\mathrm{B}}(\mathbf{x},1)^{\intercal}\Psi_{\mathrm{B}}(\mathbf{z},0)+\\ & \Psi_{\mathrm{A}}(\mathbf{x},0)^{\intercal}\Psi_{\mathrm{A}}(\mathbf{z},0)\Psi_{\mathrm{B}}(\mathbf{x},1)^{\intercal}\Psi_{\mathrm{B}}(\mathbf{z},1),\end{aligned}$$

that through simple math operations is equal to

$$\begin{aligned}\kappa_{\mathrm{A}\oplus\mathrm{B}}(\mathbf{x},\mathbf{z}) =& \kappa_{\mathrm{A}}(\mathbf{x},\mathbf{z})\kappa_{\neg\mathrm{B}}(\mathbf{x},\mathbf{z})+\\ & (\kappa_{\mathrm{A}}(\mathbf{x},\mathbf{x}) - \kappa_{\mathrm{A}}(\mathbf{x},\mathbf{z}))(\kappa_{\mathrm{B}}(\mathbf{z},\mathbf{z}) - \kappa_{\mathrm{B}}(\mathbf{x},\mathbf{z}))+\\ & (\kappa_{\mathrm{A}}(\mathbf{z},\mathbf{z}) - \kappa_{\mathrm{A}}(\mathbf{x},\mathbf{z}))(\kappa_{\mathrm{B}}(\mathbf{x},\mathbf{x}) - \kappa_{\mathrm{B}}(\mathbf{x},\mathbf{z}))+\\ & \kappa_{\neg\mathrm{A}}(\mathbf{x},\mathbf{z})\kappa_{\mathrm{B}}(\mathbf{x},\mathbf{z}),\end{aligned}$$

where $\kappa_{\neg A}$ and $\kappa_{\neg B}$ are the negation kernels applied on the domains A and B, respectively.

8.4 Propositional kernels' application

As said in the previous section, we can generate any[8] propositional formula by means of $\kappa_{A\otimes B}$ and the context-free grammar described in Definition 8.3.

In this section we describe a possible application of this framework on binary classification tasks. In these experiments we compared the propositional kernels to the linear kernel in terms of AUCs on the benchmark datasets described in Section 3.2. We used SVM as the kernel machine and we validated the C parameter in the set $\{10^{-5}, \ldots, 10^{4}\}$ and all kernels have been normalized.

Since we aimed to emphasize the expressivity power of the propositional kernels, we need to define a criterion for doing model selection. It is evident that we cannot systematically generating all possible propositional kernels and hence we decided to random generate them through the procedure described in Algorithm 8.1. In the view of a model selection step, we avoided the validation because it can be very demanding, and we opted for using the radius-margin ratio as a model selection criterion, that is supported by theoretical results as we have seen in Section 4.7.1.

In particular, for each dataset the following procedure has been employed: we generated 30 random propositional kernels using the Algorithm 8.1, and for each generated propositional kernel, the AUC and the radius-margin ratio has been calculated.

In these experiments we fixed in Algorithm 8.1 the η decay parameter to 0.7 which on average generates formulas with 5 binary operators. Figure 8.2 shows the distribution of the length of the formulas over 1000 generations by fixing η to 0.7.

By tuning η it is possible to give a bias towards shorter formulas (when $\eta > 1$) or, conversely, towards longer formulas (when $\eta \to 0$). This way of limiting the expressivity is not the only one, since the length of a formula is not the only criterion to evaluate its expressivity. For example, another criterion could consider the probability that a formula is true by considering some distribution over the variables. With this criterion, formulas with less probability of being true are more expressive than others with higher probability.

The ratio-AUC plots are depicted in Figure 8.3, where the reported AUCs are the average over a nested 5-fold cross validation (the validation only consider the C of the SVM). The datasets `monks-1` and `monks-3` have been omitted because all kernels performed with the maximum AUC. From the plots is pretty clear that choosing the kernel

[8] Any non constrained propositional formula as discussed in Section 8.3.

Algorithm 8.1: *genprop*: Algorithm for generating a random propositional kernel. K_{NOT} is a function with signature $(K_A, |A|)$ with K a propositional kernel, and K_{op} are functions with signature $(K_A, |A|, K_B, |B|)$ with K_A, K_B propositional kernels.

Input:
$\mathbf{X} \in \{0,1\}^{m \times n}$: input data (examples on the rows);
$\eta \in [0, +\infty]$: decay;
$t \in \mathbb{N}^+$: tree depth (default: 0)
Output:
K: propositional kernel;
dim: dimension of the feature space

$p \leftarrow \frac{e^{\eta t}}{1+e^{\eta t}}$
if *random(0,1)* < p **then**
  return $\mathbf{X}\mathbf{X}^\intercal, n$ ▷ `recursion base case: linear kernel`
else
  $op \leftarrow$ random($\{\neg, \wedge, \vee, \oplus, \rightarrow, \leftarrow, \leftrightarrow, \downarrow, \uparrow, \nrightarrow, \nleftarrow\}$)
  if op $= \neg$ **then**
    return $K_{NOT}(genprop(\mathbf{X}, \eta, t+1))$
  else
    return $K_{op}(genprop(\mathbf{X}, \eta, t+1), genprop(\mathbf{X}, \eta, t+1))$
  end
end

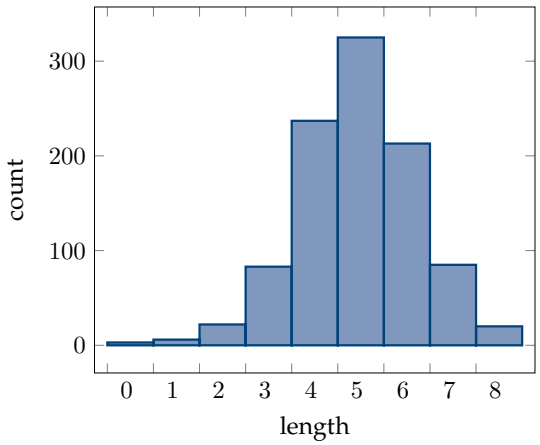


Figure 8.2: Distribution of the formulas' length on 1000 random propositional kernel generations with $\eta = 0.7$.

with minimum ratio is a very good heuristic, and hence, we have a theoretical founded criterion for avoiding validation which is in general a very demanding step.

In almost all datasets the linear kernel had very bad performance with respect to

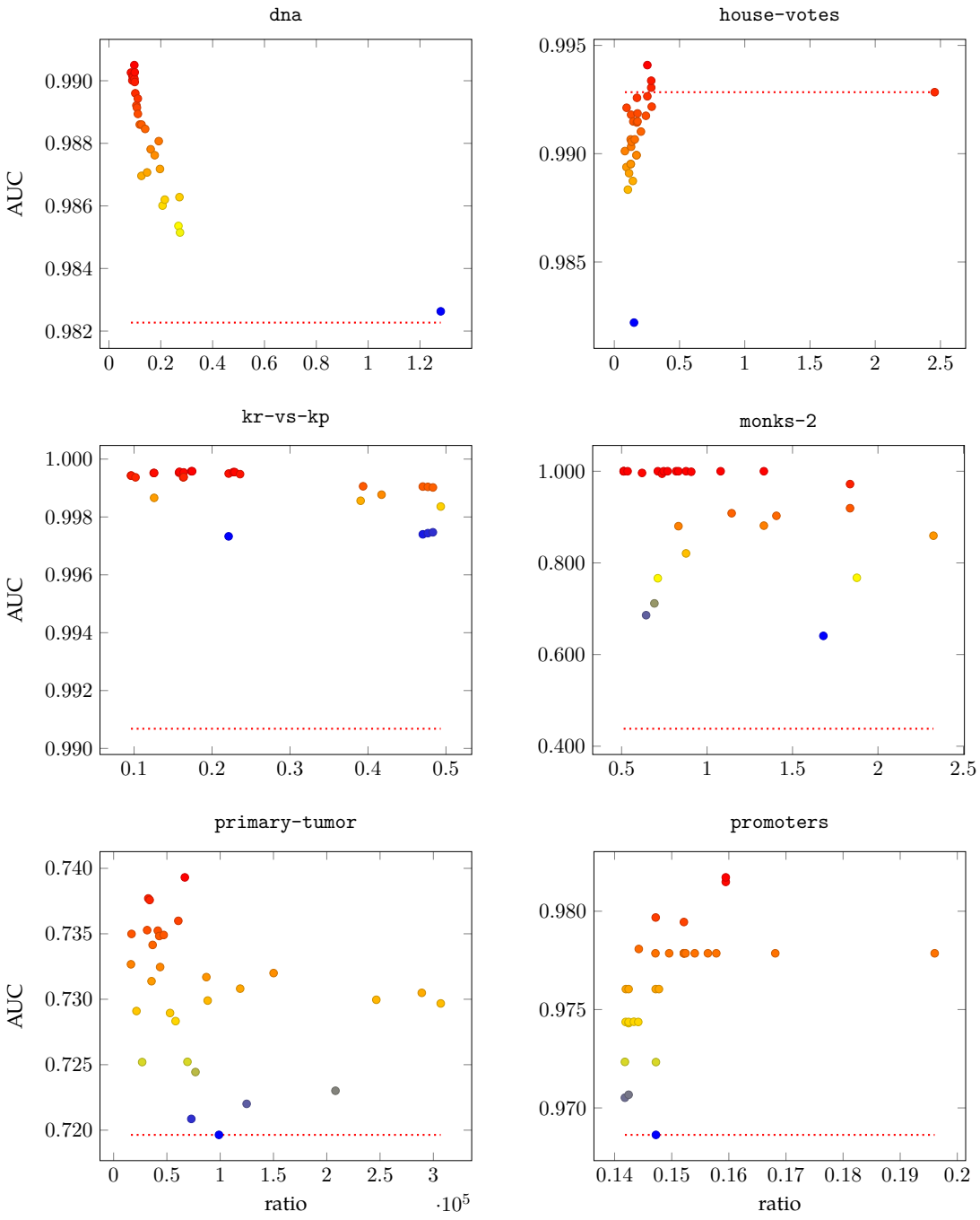


Figure 8.3: AUC score and reached ratio for each considered dataset. In general, high AUC values correspond to small ratio. The red dotted line indicates the performance of the linear kernel.

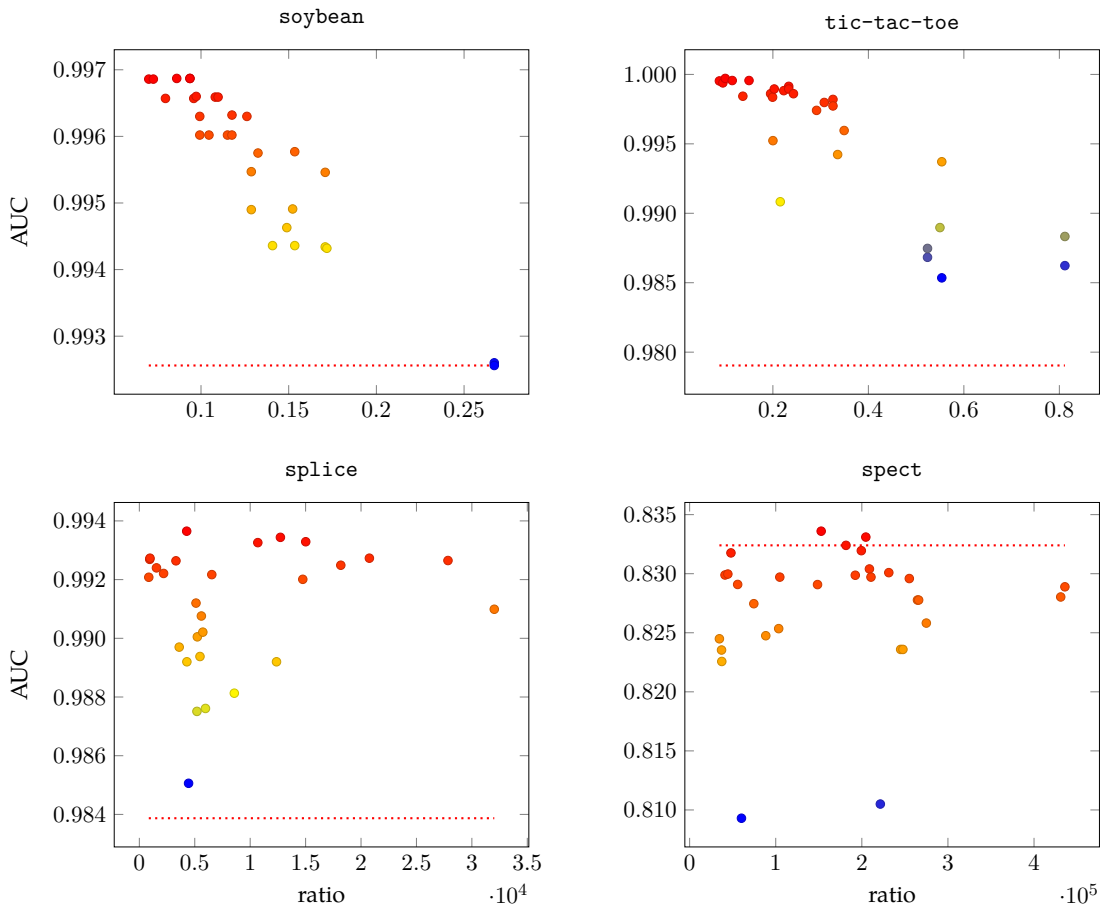


Figure 8.3: AUC score and reached ratio for each considered dataset. In general, high AUC values correspond to small ratio. The red dotted line indicates the performance of the linear kernel.

the propositional kernels. It is also noteworthy that in all datasets, but `primary-tumor`, `promoters` and `soybean`, the radius-margin ratio achieved by the linear kernel was out of scale w.r.t. to the ones in the plots. This is due to the fact that such datasets are not linearly separable. Surprisingly, despite this issue, the linear kernel performed well in `spect`.

Although this has been a feasibility test for the applicability of the propositional kernel framework, it has shown the potential of this family of kernels.

9

Boolean kernel learning

I'm afraid that the following syllogism may be used by some in the future.

Turing believes machines think;
Turing lies with men;
Therefore machines do not think.

Yours in distress, Alan.

Alan Turing

In short

In Section 4.7 we gave a general view of the Multiple Kernel Learning paradigm. In this section we present an MKL approach based on the solid theoretical concept concerning the margin and the radius of the MEB (see Section 4.5 and Section 4.7.1). Specifically, the proposed MKL algorithm, dubbed GRAM, try to minimize the exact radius-margin ratio by means of a two-phases approach: in the first phase, the kernel combination is fixed, and the optimization problems for both the radius and margin are solved; in the second phase, a gradient descent step is performed over the combination weights. We then apply this algorithm in order to learn non-parametrically the best combination of mC-kernels (that we demonstrated of having a strict relation with the family of dot-product kernels) on several binary classification tasks. A thorough empirical evaluation shows that GRAM is able to achieve state-of-the-art performance.

9.1 Dot-product kernels as combination of mC-kernels

The concept of kernels' combination is intrinsically present inside the notion of Dot-Product kernel (DPK), in fact, it has been proved [DA16, Sch42] that any kernel function of the form $\kappa(\mathbf{x}, \mathbf{z}) = f(\langle \mathbf{x}, \mathbf{z} \rangle)$, which is function only of the dot-product between

the examples, can be decomposed as a Dot-Product Polynomial (DPP), that is a linear non-negative combination of Homogeneous Polynomial Kernels (HP-kernels), as in the following:

$$\kappa(\mathbf{x},\mathbf{z}) = f(\langle \mathbf{x},\mathbf{z}\rangle) = \sum_{d=0}^{+\infty} a_d \kappa_{\text{HP}}^d(\mathbf{x},\mathbf{z}) = \sum_{d=0}^{+\infty} a_d \langle \mathbf{x},\mathbf{z}\rangle^d, \tag{9.1}$$

with opportune coefficients $a_d \geq 0$.

In both binary and multi-class contexts, the combination (9.1) can be made non-parametric by optimizing the coefficients a_d from data via MKL [DA16, LDA17].

In this section we show an extension of the above-mentioned result in the case of binary valued data $\mathbf{x},\mathbf{z} \in \{0,1\}^n$. Specifically, we demonstrate that any DPK defined on Boolean input vectors can be seen as a non-negative linear combination of mC-kernels of different degrees. We will refer to this combination as monotone Conjunctive kernel Polynomial (mCKP).

The feature space of an homogeneous polynomial kernel of a given degree d is composed by all the monomials of degree d, each weighted by some coefficient. When the input patterns are binary, many of these monomials collide in a single one, since the factors of the monomials $\mathbf{x}_i^p$ has the same value for every $p \geq 1$. This observation allows us to give the following theorems concerning the relationship between mC-kernels and HP-kernels.

Theorem 9.1

Given $\mathbf{x},\mathbf{z} \in \{0,1\}^n$, then any HP-kernel can be decomposed as a finite non-negative linear combination of mC-kernels (a mCKP) of the form:

$$\kappa_{\text{HP}}^d(\mathbf{x},\mathbf{z}) = \sum_{s=0}^{d} h(s,d)\ \kappa_{\text{mC}}^s(\mathbf{x},\mathbf{z}), \qquad h(s,d) \geq 0. \tag{9.2}$$

Proof. Given $\mathbf{x},\mathbf{z} \in \{0,1\}^n$, by definition:

$$\kappa_{\text{mC}}^s(\mathbf{x},\mathbf{z}) = \binom{\langle \mathbf{x},\mathbf{z}\rangle}{s} = \sum_{\mathbf{b}\in\mathbb{B}_s} \mathbf{x}^{\mathbf{b}}\mathbf{z}^{\mathbf{b}}$$

where $\mathbb{B}_s \equiv \{\mathbf{b} \in \{0,1\}^n \,\big|\, \|\mathbf{b}\|_1 = s\}$. Moreover, we have:

$$\begin{aligned}\kappa^d_{\text{HP}}(\mathbf{x},\mathbf{z}) = \langle \mathbf{x},\mathbf{z}\rangle^d = \left(\sum_{i=1}^n \mathbf{x}_i\mathbf{z}_i\right)^d &= \sum_{\mathbf{p}\in\mathbb{P}_d} \underbrace{\left(d!\prod_{p_i\in\mathbf{p}}\frac{1}{p_i!}\right)}_{q(\mathbf{p},d)} \mathbf{x}^{\mathbf{p}}\mathbf{z}^{\mathbf{p}} \\ &= \sum_{\mathbf{p}\in\mathbb{P}_d} q(\mathbf{p},d)\mathbf{x}^{\mathbf{p}}\mathbf{z}^{\mathbf{p}},\end{aligned} \tag{9.3}$$

with $\mathbb{P}_d \equiv \{\mathbf{p} \in \mathbb{N}_0^n \,\big|\, \|\mathbf{p}\|_1 = d\}$. By partitioning the elements $\mathbf{p} \in \mathbb{P}_d$ so that vectors with the same number of non zero entries lie in the same set, we can rewrite Eq. 9.3 as

$$\kappa^d_{\text{HP}}(\mathbf{x},\mathbf{z}) = \sum_{s=0}^d \sum_{\mathbf{p}\in\mathbb{P}_d^s} q(\mathbf{p},d)\mathbf{x}^{\mathbf{p}}\mathbf{z}^{\mathbf{p}}, \tag{9.4}$$

where $\mathbb{P}_d^s \equiv \{\mathbf{p} \in \mathbb{P}_d \,\big|\, \sum_{i=1}^n [\![p_i > 0]\!] = s\}$.

Let us now further partition the set $\mathbb{P}_d^s$ in such a way to have two vectors taken from $\mathbb{P}_d^s$ in the same class of equivalence if and only if they share the same components greater than zero. Specifically, given $\mathbf{b} \in \mathbb{B}_s$, then $\mathbb{P}_d^s(\mathbf{b}) \equiv \{\mathbf{p} \in \mathbb{P}_d^s \,|\, \forall i : p_i > 0 \iff b_i = 1\}$. With this notation, we can rewrite Eq. 9.4 as:

$$\kappa^d_{\text{HP}}(\mathbf{x},\mathbf{z}) = \sum_{s=0}^d \sum_{\mathbf{b}\in\mathbb{B}_s} \mathbf{x}^{\mathbf{b}}\mathbf{z}^{\mathbf{b}} \sum_{\mathbf{p}\in\mathbb{P}_d^s(\mathbf{b})} q(\mathbf{p},d). \tag{9.5}$$

Now, we can observe that, when s is fixed, then $\sum_{\mathbf{p}\in\mathbb{P}_d^s(\mathbf{b})} q(\mathbf{p},d)$ is constant over the elements $\mathbf{b} \in \mathbb{B}_s$. This is because the terms of the summations are the same (possibly permuted).

So, by taking any representative $\mathbf{b}_s \in \mathbb{B}_s$, we can rewrite Eq. 9.5 as:

$$\kappa^d_{\text{HP}}(\mathbf{x},\mathbf{z}) = \sum_{s=0}^d \underbrace{\left(\sum_{\mathbf{p}\in\mathbb{P}_d^s(\mathbf{b}_s)} q(\mathbf{p},d)\right)}_{h(s,d)} \left(\sum_{\mathbf{b}\in\mathbb{B}_s} \mathbf{x}^{\mathbf{b}}\mathbf{z}^{\mathbf{b}}\right) = \sum_{s=0}^d h(s,d)\ \kappa^s_{\text{mC}}(\mathbf{x},\mathbf{z}).$$

□

In the following we will show that, assuming Boolean input vectors with the same number of true variables (i.e., with fixed L^1-norm), a similar result of Theorem 9.1 holds when using normalized mC-kernels.

Theorem 9.2

Given $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$ such that $\|\mathbf{x}\|_1 = \|\mathbf{z}\|_1 = m$, $m > 0$, then any HP-kernel can be decomposed as a finite non-negative linear combination of normalized mC-kernels, that is:

$$\kappa^d_{\text{HP}}(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} h(m, s, d)\, \widetilde{\kappa}^s_{\text{mC}}(\mathbf{x}, \mathbf{z}), \qquad h(m, s, d) \geq 0. \tag{9.6}$$

where $h(m, s, d) \geq 0$ is a non-negative real value which depends on m, s and d.

Proof. Consider the normalized mC-kernel, defined as follows:

$$\widetilde{\kappa}^s_{\text{mC}}(\mathbf{x}, \mathbf{z}) = \frac{\binom{\langle \mathbf{x}, \mathbf{z} \rangle}{s}}{\binom{\langle \mathbf{x}, \mathbf{x} \rangle}{s}^{\frac{1}{2}} \binom{\langle \mathbf{z}, \mathbf{z} \rangle}{s}^{\frac{1}{2}}}.$$

Since we assume $\|\mathbf{x}\|_1 = \|\mathbf{z}\|_1 = m$, we can write:

$$\widetilde{\kappa}^s_{\text{mC}}(\mathbf{x}, \mathbf{z}) = \frac{\binom{\langle \mathbf{x}, \mathbf{z} \rangle}{s}}{\binom{m}{s}^{\frac{1}{2}} \binom{m}{s}^{\frac{1}{2}}} = \frac{1}{\binom{m}{s}} \kappa^s_{\text{mC}}(\mathbf{x}, \mathbf{z})$$

where we used the fact that for binary vectors $\|\cdot\|_1 = \|\cdot\|_2^2$ always holds and hence, by Theorem 9.1 we can conclude:

$$\kappa^d_{\text{HP}}(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} \underbrace{h(s, d) \binom{m}{s}}_{h(m,s,d)} \widetilde{\kappa}^s_{\text{mC}}(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} h(m, s, d)\, \widetilde{\kappa}^s_{\text{mC}}(\mathbf{x}, \mathbf{z}).$$

□

It is worth to notice that, when dealing with categorical data, fixing the L^1-norm of the input vectors is not so restrictive since such fixed number of active variables can be achieved by means of the one-hot encoding (without any loss of information).

Exploiting the result in Equation (9.1) and the previous theorems, we can get the following corollary.

Corollary 9.1

Given $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$ such that $\|\mathbf{x}\|_1 = \|\mathbf{z}\|_1 = m$, $m > 0$, then any DPK can be decomposed as a finite non-negative linear combination of normalized mC-kernels:

$$\kappa(\mathbf{x}, \mathbf{z}) = f(\langle \mathbf{x}, \mathbf{z} \rangle) = \sum_{s=0}^{m} g(m, s)\ \widetilde{\kappa}^{s}_{\text{mC}}(\mathbf{x}, \mathbf{z}), \quad g(m, s) \geq 0 \tag{9.7}$$

Proof. By using Theorem 9.1, if $\|\mathbf{x}\|_1 = \|\mathbf{z}\|_1 = m$, we have that $\kappa^{d}_{\text{HP}}(\mathbf{x}, \mathbf{z})$ can be seen as a non-negative linear combination of the first d normalized mC-kernels. By inserting this result in [DA16] we obtain:

$$\kappa(\mathbf{x}, \mathbf{z}) = f(\langle \mathbf{x}, \mathbf{z} \rangle) = \sum_{d=0}^{+\infty} a_d \sum_{s=0}^{d} h(m, s, d)\ \widetilde{\kappa}^{s}_{\text{mC}}(\mathbf{x}, \mathbf{z}).$$

Recalling that anytime $s > m$ then $h(m, s, d) = 0$, we can limit the inner summation up to $\min(m, d)$ as in the following

$$f(\langle \mathbf{x}, \mathbf{z} \rangle) = \sum_{d=0}^{+\infty} a_d \sum_{s=0}^{\min(d,m)} h(m, s, d)\ \widetilde{\kappa}^{s}_{\text{mC}}(\mathbf{x}, \mathbf{z}).$$

Let us now define the function

$$\hat{h}(m, s, d) = \begin{cases} h(m, s, d) & \text{if } s \leq d \\ 0 & \text{otherwise} \end{cases}.$$

By inserting it in the previous equation we get

$$f(\langle \mathbf{x}, \mathbf{z} \rangle) = \sum_{d=0}^{+\infty} a_d \sum_{s=0}^{m} \hat{h}(m, s, d)\ \widetilde{\kappa}^{s}_{\text{mC}}(\mathbf{x}, \mathbf{z}), \tag{9.8}$$

and since the inner summation does not depend on d anymore, we can rewrite (9.8) as

$$f(\langle \mathbf{x}, \mathbf{z} \rangle) = \sum_{s=0}^{m} \underbrace{\sum_{d=0}^{+\infty} a_d \hat{h}(m, s, d)}_{g(m,s)}\ \widetilde{\kappa}^{s}_{\text{mC}}(\mathbf{x}, \mathbf{z}),$$

with both a_d and $\hat{h}(m, s, d)$ non negatives, and thus we can conclude that $g(m, s) \geq 0$, which proves the theorem. □

9.2 GRAM: Gradient-based RAtio Minimization algorithm

In the previous section we showed that any Dot-Product kernel defined on Boolean vectors can be seen as a parametric linear In this section, we propose a non-parametric version of such combination by learning the coefficients of the mCKP via the optimization of the radius-margin ratio of the combined kernel. Specifically, we search on the kernel space

$$\kappa(\mathbf{x}, \mathbf{z}) = \sum_{s=1}^{P} \boldsymbol{\mu}_s \widetilde{\kappa}^s_{\text{mC}}(\mathbf{x}, \mathbf{z}), \tag{9.9}$$

where $\forall\ s, \boldsymbol{\mu}_s \geq 0$ are the parameters to optimize with the constraint that $\sum_{s=1}^{P} \boldsymbol{\mu}_s = 1$. Inside this space we want to find the kernel that minimizes the exact (squared) radius-margin ratio for which we have already shown being an upper bound of the leave-one-out error (see Section 4.7.1).

Since we are considering convex kernel combinations, in order to guarantee that the solution $\boldsymbol{\mu} = [\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_P]$ stays in the feasible range of values, i.e., $\|\boldsymbol{\mu}\| = 1$, we exploit the properties of the exponential, and we perform a change of variables by introducing a new vector of variables $\boldsymbol{\beta} \in \mathbb{R}^P$ such that

$$\boldsymbol{\mu}_s(\boldsymbol{\beta}) = \frac{e^{\boldsymbol{\beta}_s}}{\sum\limits_{r=1}^{P} e^{\boldsymbol{\beta}_r}}, \qquad \forall\ s \in [P].$$

This allows us to obtain an unconstrained problem easier to optimize.

First of all, let us recall how the radius of the MEB and the margin are computed. Given a normalized kernel matrix $\widetilde{\mathbf{K}}$ constructed over examples of the dataset $\mathcal{D} \equiv \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, and the diagonal matrix $\mathbf{Y} \in \{0, 1, -1\}^L$, s.t. $\mathbf{Y}_{i,i} = y_i$, then the radius of the MEB and the margin are computed by

$$\rho^2 = \min_{\boldsymbol{\gamma} \in \Gamma}\ \boldsymbol{\gamma}^\intercal \mathbf{Y} \widetilde{\mathbf{K}} \mathbf{Y} \boldsymbol{\gamma}, \qquad R^2 = 1 - \min_{\boldsymbol{\alpha} \in \mathcal{A}} \boldsymbol{\alpha}^\intercal \widetilde{\mathbf{K}} \boldsymbol{\alpha},$$

where Γ is defined as in Section 4.3.3, and $\|\boldsymbol{\alpha}\|_1 = 1$. For later convenience, let us now define the combined kernel as a function of $\boldsymbol{\beta}$ as

$$\widetilde{\mathbf{K}}_{\boldsymbol{\beta}} = \sum_{r=1}^{P} \boldsymbol{\mu}_r(\boldsymbol{\beta}) \widetilde{\mathbf{K}}_r, \tag{9.10}$$

where $\widetilde{\mathbf{K}}_r$ are the normalized kernel matrix associated with the r-th kernel function κ_r.

We can now write the radius-margin ratio minimization problem as in the following:

$$\begin{aligned}\min_{\boldsymbol{\beta}} \Psi(\boldsymbol{\beta}), \text{ with } \Psi(\boldsymbol{\beta}) &= \frac{1-\hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})^\intercal \left(\sum_{r=1}^{P} \boldsymbol{\mu}_r(\boldsymbol{\beta})\widetilde{\mathbf{K}}_r\right)\hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})}{\hat{\boldsymbol{\gamma}}(\boldsymbol{\beta})^\intercal \mathbf{Y}\left(\sum_{r=1}^{P} \boldsymbol{\mu}_r(\boldsymbol{\beta})\widetilde{\mathbf{K}}_r\right)\mathbf{Y}\hat{\boldsymbol{\gamma}}(\boldsymbol{\beta})} \\ &= \frac{1-\hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})^\intercal \widetilde{\mathbf{K}}_{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})}{\hat{\boldsymbol{\gamma}}^\intercal \mathbf{Y}\widetilde{\mathbf{K}}_{\boldsymbol{\beta}}\mathbf{Y}\hat{\boldsymbol{\gamma}}(\boldsymbol{\beta})},\end{aligned} \tag{9.11}$$

where

$$\hat{\boldsymbol{\alpha}}(\boldsymbol{\beta}) = \arg\min_{\boldsymbol{\alpha}\in\mathcal{A}} \boldsymbol{\alpha}^\intercal \widetilde{\mathbf{K}}_{\boldsymbol{\beta}}\boldsymbol{\alpha}, \tag{9.12}$$

with $\mathcal{A} \equiv \{\boldsymbol{\alpha} \in \mathbb{R}_+^L, \|\boldsymbol{\alpha}\|_1 = 1\}$, and

$$\hat{\boldsymbol{\gamma}}(\boldsymbol{\beta}) = \arg\min_{\boldsymbol{\gamma}\in\Gamma} \boldsymbol{\gamma}^\intercal \mathbf{Y}\widetilde{\mathbf{K}}_{\boldsymbol{\beta}}\mathbf{Y}\boldsymbol{\gamma}, \tag{9.13}$$

with $\Gamma \equiv \{\boldsymbol{\gamma} \in [0,1]^L \mid \sum_{j:y_j=1} \boldsymbol{\gamma}_i = 1, \sum_{i:y_i=0} \boldsymbol{\gamma}_j = 1\}$. By definition $\sum_{r=1}^{P} \boldsymbol{\mu}_r(\boldsymbol{\beta}) = 1$, thus

$$\Psi(\boldsymbol{\beta}) = \frac{\sum_{r=1}^{P} e^{\boldsymbol{\beta}_r} \overbrace{\left(1-\hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})^\intercal \widetilde{\mathbf{K}}_r \hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})\right)}^{\mathbf{a}_r(\boldsymbol{\beta})}}{\sum_{r=1}^{P} e^{\boldsymbol{\beta}_r} \underbrace{\left(\hat{\boldsymbol{\gamma}}(\boldsymbol{\beta})^\intercal \mathbf{Y}\widetilde{\mathbf{K}}_r \mathbf{Y}\hat{\boldsymbol{\gamma}}(\boldsymbol{\beta})\right)}_{\mathbf{b}_r(\boldsymbol{\beta})}} \approx \frac{\langle e^{\boldsymbol{\beta}}, \mathbf{a}\rangle}{\langle e^{\boldsymbol{\beta}}, \mathbf{b}\rangle} = \widehat{\Psi}(\boldsymbol{\beta}) \tag{9.14}$$

where $e^{\boldsymbol{\beta}} = [e^{\boldsymbol{\beta}_1}, \ldots, e^{\boldsymbol{\beta}_P}]$, $\mathbf{a} = [a_1, a_2, \ldots, a_P]^\intercal$, $\mathbf{b} = [b_1, b_2, \ldots, b_P]^\intercal$ and we assume $\mathbf{a}, \mathbf{b}$ constants around a given $\boldsymbol{\beta}$. In order to optimize the function $\Psi(\boldsymbol{\beta})$ we perform a series of steps of gradient descent on the approximated function $\widehat{\Psi}(\boldsymbol{\beta})$ followed by a new computation of both $\mathbf{a} = \mathbf{a}(\boldsymbol{\beta})$ and $\mathbf{b} = \mathbf{b}(\boldsymbol{\beta})$.

The gradient of $\bar{\Psi}(\boldsymbol{\beta})$ with respect to the parameter $\boldsymbol{\beta}_r, \forall r \in [P]$ is computed by:

$$\frac{\partial\widehat{\Psi}(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}_r} = \frac{\mathbf{a}_r e^{\boldsymbol{\beta}_r}\langle e^{\boldsymbol{\beta}}, \mathbf{b}\rangle - \mathbf{b}_r e^{\boldsymbol{\beta}_r}\langle e^{\boldsymbol{\beta}}, \mathbf{a}\rangle}{\langle e^{\boldsymbol{\beta}}, \mathbf{b}\rangle^2} = \frac{e^{\boldsymbol{\beta}_r}(\mathbf{a}_r\langle e^{\boldsymbol{\beta}}, \mathbf{b}\rangle - \mathbf{b}_r\langle e^{\boldsymbol{\beta}}, \mathbf{a}\rangle)}{\langle e^{\boldsymbol{\beta}}, \mathbf{b}\rangle^2}$$

To summarize: starting from $\boldsymbol{\beta} = \mathbf{0}$, and consequently from the uniform distribution over base kernels $\boldsymbol{\mu}(\boldsymbol{\beta})$, at each iteration, the kernel combination is computed using the current $\boldsymbol{\mu}(\boldsymbol{\beta})$, and then the vectors $\mathbf{a} = \mathbf{a}(\boldsymbol{\beta})$ and $\mathbf{b} = \mathbf{b}(\boldsymbol{\beta})$ are computed as described

above. Finally, the update of $\boldsymbol{\beta}$ and $\boldsymbol{\mu}(\boldsymbol{\beta})$ are performed as in the following:

$$\boldsymbol{\beta}_r \leftarrow \boldsymbol{\beta}_r - \eta \frac{e^{\boldsymbol{\beta}_r} \sum_s e^{\boldsymbol{\beta}_s} (\mathbf{a}_r \mathbf{b}_s - \mathbf{a}_s \mathbf{b}_r)}{\langle e^{\boldsymbol{\beta}}, \mathbf{b} \rangle^2} \quad \forall r \in [P], \tag{9.15}$$

$$\boldsymbol{\mu} \leftarrow \frac{1}{\sum_r e^{\boldsymbol{\beta}_r}} e^{\boldsymbol{\beta}}, \tag{9.16}$$

where η is the learning rate. This iterative procedure continues until one of the stop criterion is met.

Algorithm 9.1: GRAM

Input:
$\mathbf{KL} = [\widetilde{\mathbf{K}}_1, \widetilde{\mathbf{K}}_2, \ldots, \widetilde{\mathbf{K}}_P]$: weak kernels' list;
$\mathbf{Y} \in \{-1, 0, 1\}^{l \times l}$: diagonal matrix of labels;
η: initial learning rate
Output: $\boldsymbol{\mu} \in \mathbb{R}^P_{\geq 0}$: vector of kernels' weights

$\boldsymbol{\beta} \leftarrow [0, 0, \ldots, 0] \in \mathbb{R}^P$
$\boldsymbol{\mu} \leftarrow e^{\boldsymbol{\beta}} \,/P$
$t \leftarrow 0$
$\eta^{(0)} \leftarrow \eta$
do
  $\mathbf{K}^{(t)} \leftarrow \sum_{r=1}^{P} \mu_r \widetilde{\mathbf{K}}_r$
  $\boldsymbol{\alpha}^{(t)} \leftarrow \mathit{argradius}(\widetilde{\mathbf{K}}^{(t)})$
  $\boldsymbol{\gamma}^{(t)} \leftarrow \mathit{argmargin}(\widetilde{\mathbf{K}}^{(t)}, \mathbf{Y})$
  $\mathbf{a} \leftarrow (1 - \boldsymbol{\alpha}^{(t)\intercal} \widetilde{\mathbf{K}}_r \boldsymbol{\alpha}^{(t)}) \quad \forall \widetilde{\mathbf{K}}_r \in \mathbf{KL}$
  $\mathbf{b} \leftarrow (\boldsymbol{\gamma}^{(t)\intercal} \mathbf{Y} \widetilde{\mathbf{K}}_r \mathbf{Y} \boldsymbol{\gamma}^{(t)}) \quad \forall \widetilde{\mathbf{K}}_r \in \mathbf{KL}$
  $\Delta\boldsymbol{\beta} \leftarrow \frac{e^{\beta_r} \sum_s e^{\beta_s} (a_r b_s - a_s b_r)}{\langle e^{\boldsymbol{\beta}}, \mathbf{b} \rangle^2} \quad \forall\, r \in [1, P]$
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \eta^{(t)} \cdot \Delta\boldsymbol{\beta}$
  $\boldsymbol{\mu} \leftarrow \frac{e^{\boldsymbol{\beta}}}{\sum_r e^{\beta_r}}$
  $t \leftarrow t + 1$
  $\eta^{(t)} \leftarrow \eta^{(t-1)}$ if the current step improved the solution, else $\frac{\eta^{(t-1)}}{2}$
while *stop conditions are not reached*
return $\boldsymbol{\mu}$

A summary of the whole procedure is reported in Algorithm 9.1. In the algorithm, the functions *radius* and *margin* refer to the Equation (4.32) and (4.31), respectively.

It is worth to notice that GRAM is not strictly related with the Boolean kernels since its optimization criterion is always valid and thus it is applicable with all kind of kernels.

9.2.1 Stopping criteria and convergence

The convergence is reached as soon as one of the following stopping criteria is met:

- maximum number of iterations (*max_iter*) is reached;
- the improvement of the objective function is below an a-priori fixed tolerance.

Concerning the selection of the gradient step η, a procedure similar to the well known *Backtracking Line-Search* [BV04] has been used. Starting from $\eta = 1$, anytime the value of the objective function is worse (i.e., greater) than its value in the previous iteration, i.e., $\widehat{\Psi}(\boldsymbol{\beta})^{(i+1)} \geq \widehat{\Psi}(\boldsymbol{\beta})^{(i)}$, than the gradient step is updated according to $\eta^{(i+1)} \leftarrow \frac{1}{2}\eta^{(i)}$ and the procedure is repeated without updating the weights $\boldsymbol{\mu}^{(i+1)} \leftarrow \boldsymbol{\mu}^{(i)}$.

9.2.2 Extension to non-normalized kernels

In the algorithm described above we only considers normalized kernels, however it is possible to extend such procedure to non-normalized kernels. In order to deal with non-normalized kernels, the objective function to minimize, i.e., $\Psi(\boldsymbol{\beta})$, has to be modified as in the following

$$\Psi(\boldsymbol{\beta}) = \frac{\hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})^\intercal \left(\sum_{r=1}^{P} \boldsymbol{\mu}_r(\boldsymbol{\beta}) \boldsymbol{d}_r\right) - \hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})^\intercal \mathbf{K}_{\boldsymbol{\beta}} \hat{\boldsymbol{\alpha}}(\boldsymbol{\beta})}{\hat{\boldsymbol{\gamma}}(\boldsymbol{\beta})^\intercal \mathbf{Y} \mathbf{K}_{\boldsymbol{\beta}} \mathbf{Y} \hat{\boldsymbol{\gamma}}(\boldsymbol{\beta})}, \tag{9.17}$$

where $\mathbf{d}_r$ is a l-dimensional vector containing the diagonal of the r-th kernel matrix $\mathbf{K}_r$. The evaluation of $\mathbf{a}(\boldsymbol{\beta})$ have been modified, since the radius optimization sub-problem now considers the non-normalized case. The rest of the procedure depends on $\mathbf{a}(\boldsymbol{\beta})$ and optimizes it directly, and so it is not affected by the above modification. Nonetheless, in our evaluation setting we focused only on the normalized case. In the remainder with m we will refer to the number of ones in the input vectors, which is fixed for each dataset.

9.2.3 Computational complexity

The GRAM algorithm optimizes the radius-margin ratio by solving a two-layered procedure, as described in Section 9.2.

In each step, in order to evaluate the current ratio, the algorithm solves two convex optimization problems with linear constraints, whose complexity is more than quadratic respect to the number of examples (lines 7 and 8 of Algorithm 9.1). The computational cost of these convex problems is independent of the number of kernels.
When the optimization problems are solved, $\mathbf{a}$ and $\mathbf{b}$ are evaluated, then the gradient vector $\Delta\boldsymbol{\beta}$ and weights $\boldsymbol{\mu}$ are computed (lines 9-13). However, these operations depend linearly on the number of base kernels.
Finally, the complete procedure is repeated until convergence, and the overall computational complexity depends on the number of iterations.

On the other hand, in order to solve the problem efficiently, the algorithm requires the whole set of kernel matrices in memory, ensuring a fast access to each of them.

9.3 Evaluation

In this section the whole set of experiments is presented. The set of considered datasets are the one described in Section 3.2, with the exception of the `monks` datasets because all the MKL approaches have been able to achieve the maximum AUC. In the following we will describe the experimental methodology and we will also discuss the comparisons with several hard baselines, including state-of-art MKL algorithms.

9.3.1 The base learner

In all the following experiments, an hard margin SVM (that is KOMD with $\lambda = 0$, see Section 4.3.3) has been used as base learner to fit models exploiting kernels combined by MKL algorithms. Combinations are formed by mC-kernels with different degrees. Moreover, since we employ an hard SVM, for any combination of P kernels, an identity matrix $\mathbf{I}$ of proper dimension has been added in order to ensure the linear separability of the data. So, the considered kernels' combination has the following form

$$\widetilde{\mathbf{K}}_{\boldsymbol{\mu}} = \sum_{s=1}^{P} \boldsymbol{\mu}_s \widetilde{\mathbf{K}}_s + \boldsymbol{\mu}_{P+1}\mathbf{I} \tag{9.18}$$

where $\widetilde{\mathbf{K}}_s$ is the normalized kernel matrix computed by using the mC-kernel of arity s, and

$$\sum_{s=1}^{P+1} \boldsymbol{\mu}_s = 1, \quad \text{and} \quad \boldsymbol{\mu}_s \geq 0 \quad \forall s \in [P+1].$$

Note that, if there are not duplicated examples, the normalized mCK of arity m is already an identity kernel. In fact, for all $\mathbf{x}$, $\kappa_{\wedge}^{m}(\mathbf{x},\mathbf{x}) = \binom{m}{m} = 1$ and for $\mathbf{x} \neq \mathbf{z}$, $\kappa_{\wedge}^{m}(\mathbf{x},\mathbf{z}) = \binom{\langle \mathbf{x},\mathbf{z} \rangle}{m} = 0$ since $\langle \mathbf{x},\mathbf{z} \rangle < m$..
Learning the combination via multiple kernel learning on an hard-margin SVM is an alternative to the soft-margin L2-SVM [CKS+03], where the value of the trade-off hyper-parameter C is selected through a validation procedure, and the kernel is composed by the fist P terms of the combination, i.e., excluding the identity. For an L2-SVM the margin ρ can be efficiently computed by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b,\rho,\boldsymbol{\xi}} \; & C\|\boldsymbol{\xi}\|_2^2 - \rho \\ \text{s.t.} \; & y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b) \geq \rho - \boldsymbol{\xi}_i \qquad \forall i = 1,\ldots,l \\ & \|\mathbf{w}\|^2 = 1 \end{aligned}$$

which can be mapped into its dual form [STC04] as

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \; & -\boldsymbol{\alpha}^{\intercal}\mathbf{Y}\left(\mathbf{K}+\mu(C)\mathbf{I}\right)\mathbf{Y}\boldsymbol{\alpha} \\ \text{s.t.} \; & \|\boldsymbol{\alpha}\|_1 = 1 \\ & \boldsymbol{\alpha}^{\intercal}\mathbf{y} = 0 \\ & \boldsymbol{\alpha} \succeq 0 \end{aligned}$$

where in our case $\mathbf{K} = \sum_{s=0}^{P} \boldsymbol{\mu}_s \widetilde{\mathbf{K}}$, $\mu(C) = \boldsymbol{\mu}_{P+1}$ and $\mathbf{y}$ is the diagonal of $\mathbf{Y}$.

The risk of learning the value $\mu(C)$ via MKL, instead of through a standard validation procedure, is to overfit the data if the MKL algorithm has weak regularization capability. For this reason, in our experiments the MKL approaches are also compared in terms of capacity of regularization and overfit resistance.

9.3.2 Evaluation protocol

The weak kernels considered inside the combination are all the normalized mC-kernels from the degree 1 up to m, where m is the number of active (i.e., non-zero) features in each example in a benchmark dataset.

In order to evaluate our MKL algorithm, its performances have been compared with the ones of several state-of-the-art MKL methods in terms of both AUC and radius-margin ratio (or simply ratio).

The MKL methods considered in our experiments are listed below:

Average: the average of the weak kernels, with weights $\forall r, \boldsymbol{\mu}_r = \frac{1}{P}$. Even though this might seems a trivial baseline, it is well known to work fine;

EasyMKL: EasyMKL [AD15] is a state-of-the-art MKL technique which tries to maximize the sole margin between the classes;

R-MKL: conversely to EasyMKL, this MKL approach, presented in [DKWH09], aims to minimize an approximation of the radius-margin ratio;

GRAM: the MKL algorithm presented in this chapter.

For comparison, the available data have been equally (i.e., 50/50) split in train and test sets: the training set has been used to learn the combined kernel for each MKL method. The combined kernel returned by each approach has been used to evaluate the ratio and then to fit an hard-margin SVM as base learner. Note that the radius-margin ratio considered in the evaluation is defined as

$$\text{ratio} = l^{-1}\frac{R^2}{\rho^2},$$

Dataset	GRAM	R-MKL	EasyMKL	Average
dna	**98.70**±0.24 10.35±0.33	98.36±0.28 10.83±0.36	98.30±0.27 12.26±0.32	97.89±0.29 17.76±0.19
house-votes	**99.15**±0.42 6.74±0.63	99.01±0.38 7.21±0.82	98.88±0.44 7.65±0.71	98.85±0.46 7.92±0.61
kr-vs-kp	**99.92**±0.03 7.06±0.12	99.88±0.04 7.52±0.10	99.90±0.04 7.24±0.12	99.88±0.04 7.52±0.10
primary-tumor	**72.65**±2.81 21.96±0.68	68.80±2.96 78.69±9.14	72.05±2.72 22.18±0.70	68.81±2.96 78.90±9.16
promoters	95.99±1.63 16.68±1.11	96.00±1.61 16.70±1.12	**96.30**±1.18 23.43±0.38	96.29±1.20 23.65±0.33
soybean	**99.55**±0.48 7.93±0.87	**99.55**±0.46 8.59±0.81	99.54±0.39 9.39±0.69	99.44±0.43 11.20±0.54
spect	84.00±3.04 13.63±1.13	76.97±4.98 71.46±16.95	**84.17**±3.11 14.13±1.15	77.00±4.96 71.66±16.99
splice	**99.19**±0.12 8.55±0.22	98.81±0.15 9.49±0.91	98.89±0.15 10.42±0.27	98.74±0.17 17.23±0.59
tic-tac-toe	**99.83**±0.15 12.81±0.23	99.26±0.33 15.21±0.30	99.36±0.30 14.67±0.26	99.26±0.33 15.22±0.30

Table 9.1: AUC(%) scores (first row) and ratios ($\times 10^2$, second row) of the compared MKL methods. In **bold** are highlighted the best AUCs for each benchmark dataset.

where l is the number of training examples, R the radius of the MEB and ρ the margin. Finally, the AUC scores have been calculated on the test set. In order to improve the statistical significance of the results, 30 runs of each experiment with different splits (the same set for all the methods) have been performed. Regarding the GRAM stopping criteria, the tolerance has been fixed to 10^{-8}, and the maximum number of iterations to 1000, which has never been reached in our experiments.

9.3.3 Experimental results

The mean AUC (and standard deviation) evaluated on the test sets as well as the average radius-margin (and standard deviation) ratio on the training sets are reported in Table 9.1. As expected, the results show a significant ratio improvement of the GRAM algorithm with respect to the other MKL baselines, in particular, w.r.t. EasyMKL and Average.

In order to give an empirical confirmation about the relation between the radius-margin ratio and the error achieved by a representation, a qualitative comparison has

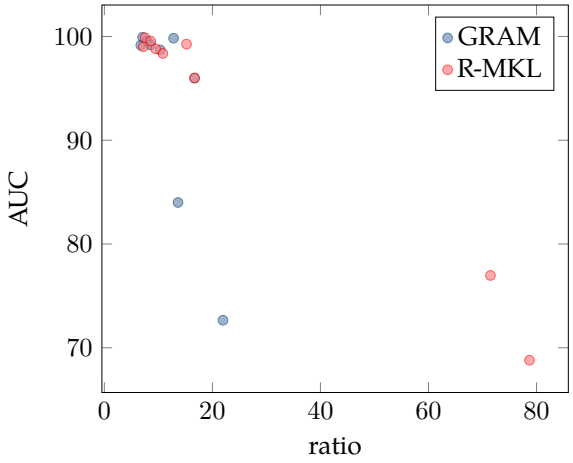


Figure 9.1: AUC score and reached ratio for each considered dataset. High AUC values correspond to small ratio.

been considered, and it is summarized in Figure 9.1. The figure concerns GRAM and the R-MKL algorithm, and it is easy to notice that high AUC scores correspond to small radius-margin ratios (upper left corner), while relatively low AUCs correspond to quite high ratios.

9.3.4 The selection of the number of kernels

In the previous experiments, $m+1$ kernels have been used to learn the best Dot-Product Kernel for a given problem, where m is fixed and it depends on the number of non-zero features for each example. However, m may not be the best choice for a problem, and algorithms may overfit when this value increases. Moreover, for some datasets, the whole set of kernels used inside the MKL combination can be very expensive to compute and to handle in memory, such as `splice`, `promoters` or `kr-vs-kp`, where the m value and the number of examples l are high.

In order to limit this issue, an extension of the previous methodology have been analyzed, in which only a subset of the whole set of m kernels has been considered. More formally, for each $d \in [m]$, we consider combinations of $d+1$ kernels including mC-kernels with degrees $1, \ldots, d$ and the identity matrix. Note that a combination of $d < m-1$ mC-kernels may bound the expressiveness of the algorithm, and the solution may not be the best DPK for the given problem, but only an approximation, according to the theorems showed in Section .

In Figure , for all datasets the AUC score and ratio reached by all MKL methods while increasing the number d of mC-kernel are reported.

As shown in the figures, the proposed algorithm always achieves better ratio scores compared with other state-of-art MKL methods. Furthermore, an empirical evidence is

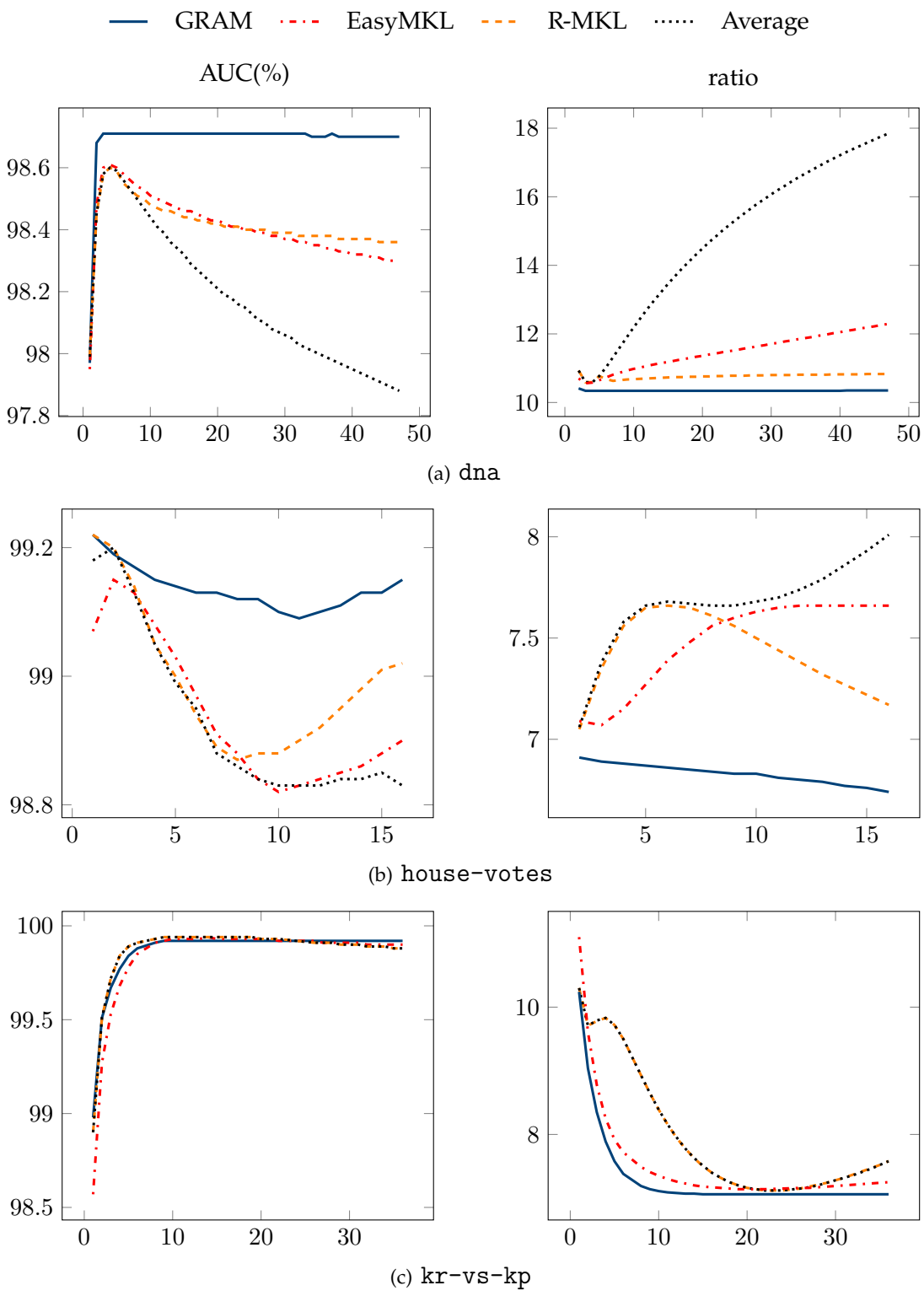
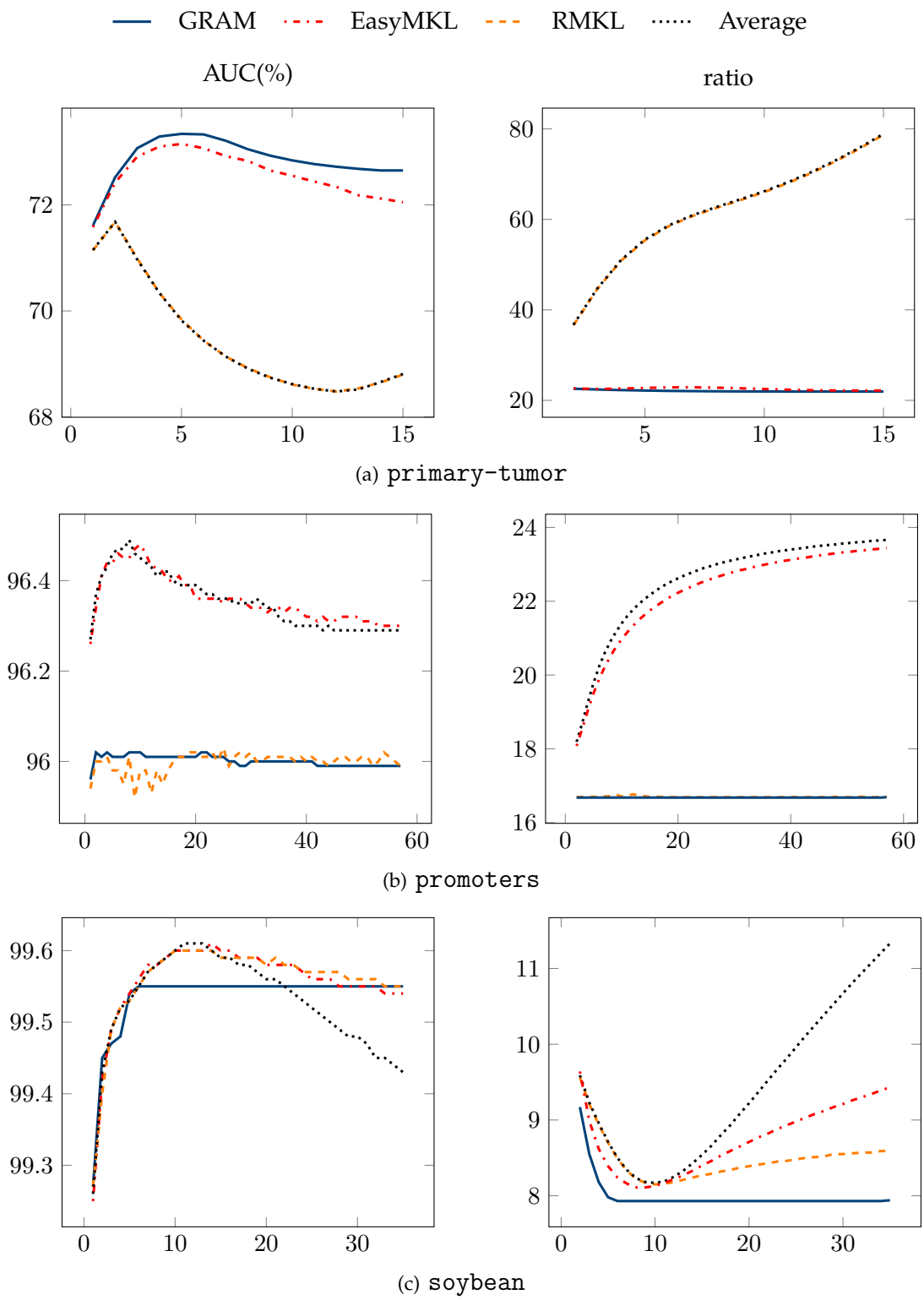


Figure 9.2: AUC (left) and ratio (right) varying the degree of the mC-kernel (x axis).



(a) `primary-tumor`

(b) `promoters`

(c) `soybean`

Figure 9.2: AUC(%) (left) and ratio (right) varying the degree of the mC-kernel (x axis).

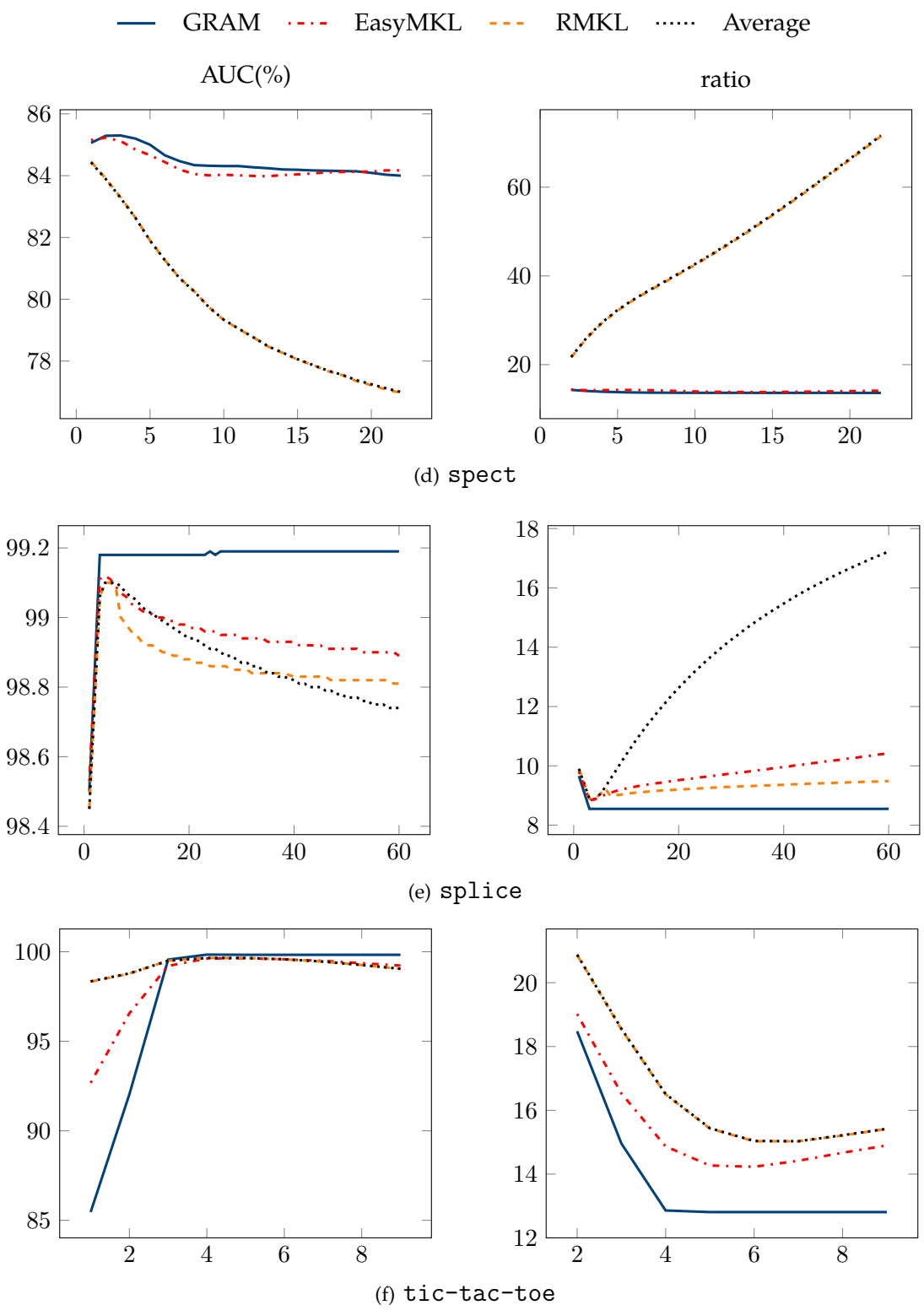


Figure 9.2: AUC(%) (left) and ratio (right) varying the degree of the mC-kernel (x axis).

that GRAM's ratio do not suffer from overfitting, and the algorithm is able to discard kernels which does not contribute in the minimization of the ratio, and hence it assigns to them a null weight. So, when a new kernel is added to the list, the algorithm can only improve its solution, reducing the ratio. This means that the number d of used kernels is not an hyperparameter and it does not require validation. However, a fixed number of kernels can be a poor choice for the other algorithms since the models may overfit easily and m may not be the best number of mCKs to use (as highlighted in the figures).

The AUC scores also remain stable in the proposed method, showing an empirical evidence of the effectiveness of the ratio-optimization approach.

Convergence and computational complexity

The computational time for a representative set of datasets is depicted in Figure 9.3. It is worth to notice that it is possible to further improve the efficiency of the quadratic programming problems by exploiting, for example, the Sequential Minimal Optimization (SMO) technique [Pla98]. Figure 9.4 shows two representative empirical evaluations

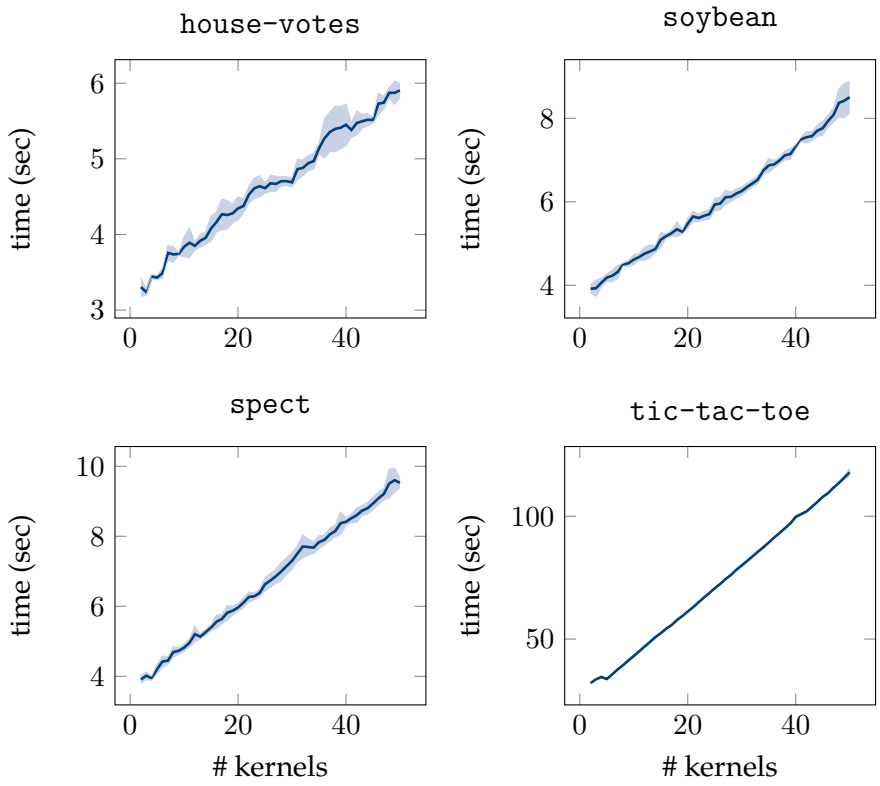


Figure 9.3: Average computational time of the algorithm using different number of weak-kernels while fixing the number of steps to 100. For each dataset we run the algorithm 5 times on an Intel ® Core™ i7-6700HQ CPU @ 2.60GHz 2.59GHz.

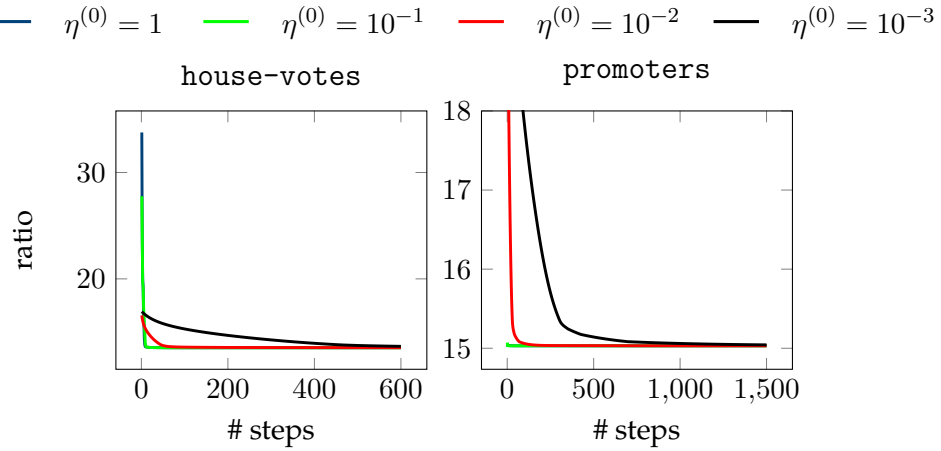


Figure 9.4: Convergence of GRAM with different initial η values.

of the GRAM's convergence by using the backtracking line-search with different initial learning rates. In both the depicted datasets we can notice that, whatever the initialization of the learning rate η, the algorithm is able to reach a minimum in a reasonable number of iterations.

Weights' evaluation

In order to better understand the behaviour of the algorithms, in Figure 9.5 is reported the weights' distribution for GRAM and the baselines.

From the figure it is self-evident that margin maximization can give very different combinations with respect to the minimization of the radius-margin ratio. We can observe that the weight vectors learned by GRAM are very sparse, and hence only a small subset of kernels are combined to form the final kernel. The most typical configuration sets only two coefficients with large values, one low degree mC-kernel and one high degree mC-kernel.

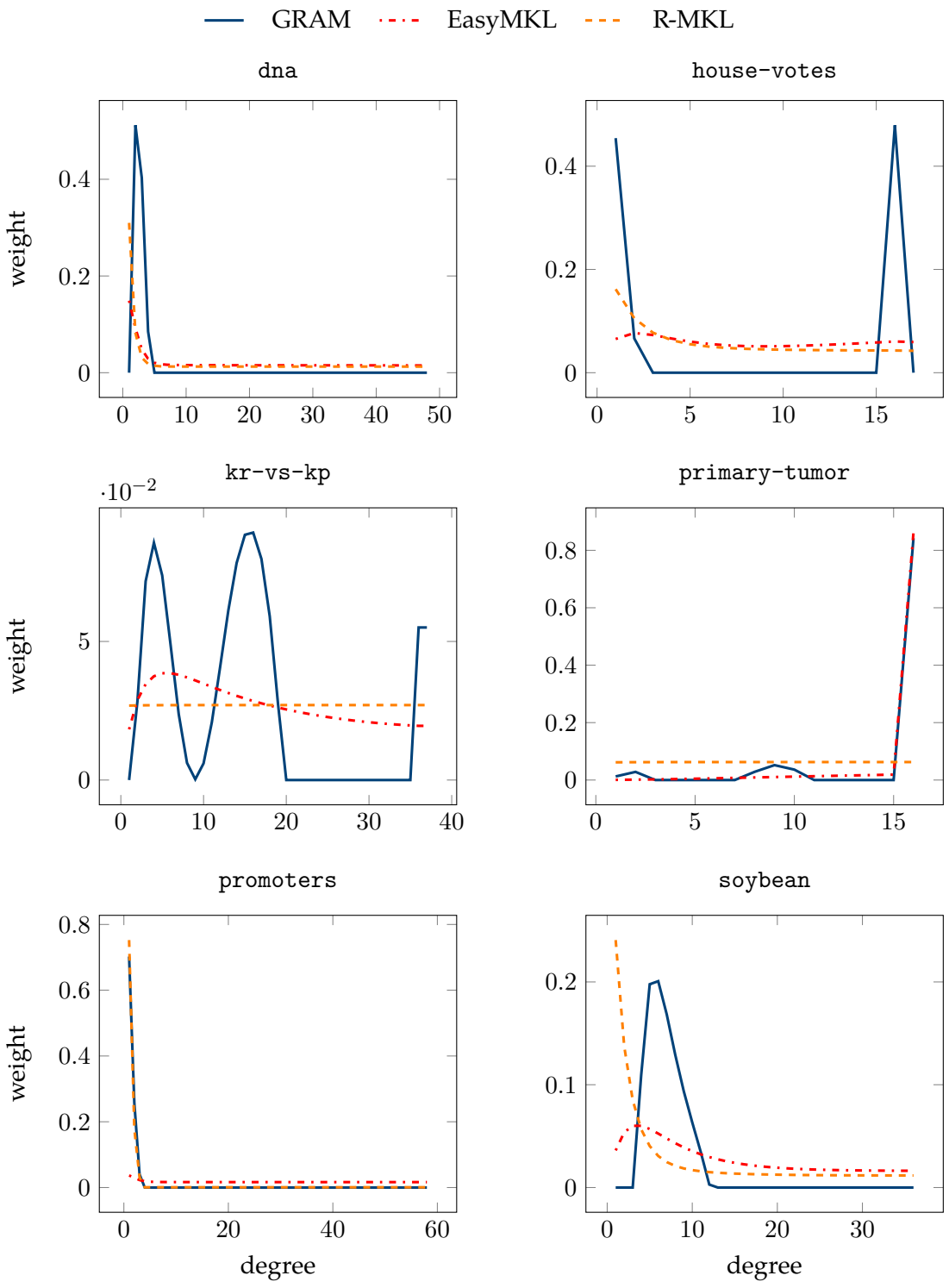


Figure 9.5: Distribution of the weights when combining mC-kernels of degrees 1 to m. For each dataset, we appended a further identity matrix to the kernels list.

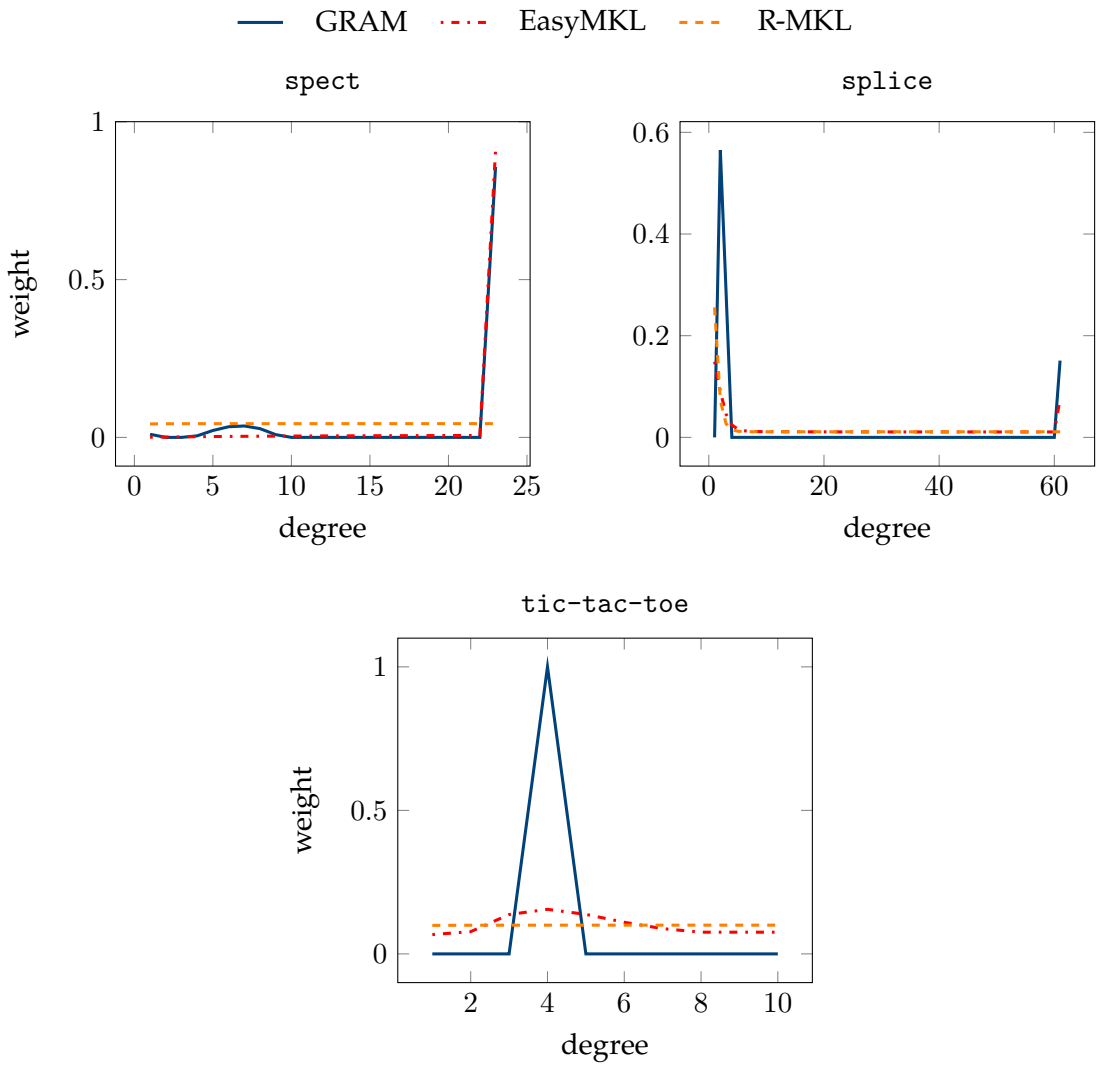


Figure 9.5: Distribution of the weights when combining mC-kernels of degrees 1 to m. For each dataset, we appended a further identity matrix to the kernels list.

PART III

Recommender Systems applications

Summary

10

Kernel-based collaborative filtering

> We are leaving the Information Age and entering the Recommendation Age. [...] Information gathering is no longer the issue - making smart decisions based on the information is now the trick.
>
> *The long tail, 2006*
> Chris Anderson

In short

In Section 2.2 and Chapter 5 we gave an overview of what recommender systems are, which are the peculiarities of different kinds of feedback, and we also presented a brief review of the state-of-the-art method for top-N item recommendation. In this chapter we present a real-world application of the Boolean kernel framework. In particular, we start by introducing a novel kernel-based method for CF top-N recommendation, which is based on the seminal method called CF-OMD described in Section 5.2. Then, such method is used with different kernels on several implicit feedback datasets. Empirical results show how Boolean kernels, in particular the mD-kernel, can improve the recommendation accuracy. Extensive analysis show how the mD-kernels are able to alleviate the sparsity issue of this kind of datasets when kernels are used. Moreover, the provided solution can be used to extract rules for explaining the recommendation, for example by using the method described in Chapter 7.

In the remainder of the chapter we will use the same notation as described in Section 2.2 and in Chapter 5.

10.1 Efficient CF-OMD

Despite CF-OMD has shown state-of-the-art performance in terms of AUC [Aio14], it is evident that such model is not suitable to deal with large datasets. In fact, let us assume that each optimization problem can be solved by an algorithm with a complexity quadratic on the number of parameters, then, since we have to solve one problem for each user, the total complexity would be $\mathcal{O}(n_{\text{TS}}m^2)$, where n_{ts} is the number of users in the test set.

So, efficiency wise, the main drawback of CF-OMD is the number of parameters it has to estimate, that is equal to the number of items. By analyzing the results reported in [Aio14], we can notice that increasing the value of λ_n do not particularly affect the achieved AUC. This can be justified by the fact that high values of λ_n tend to flatten the contribution of the ambiguous (implicit) negative feedbacks toward the average, and this mitigate the relevance of these noisy information.

In collaborative filtering contexts the data sparsity is particularly accentuate, this means that, on average, the number of ambiguous negative feedbacks is orders of magnitude greater than the number of positive ones. More formally, given a user u, let $m_\oplus = |\mathcal{I}_u|$ and $m_\ominus = |\mathcal{I} \setminus \mathcal{I}_u|$ then $m = m_\ominus + m_\oplus$, where $m_u^+ \ll m_\ominus$, and generally $\mathcal{O}(m) = \mathcal{O}(m_\ominus)$.

By exploiting these last observations, we can simplify the optimization problem (5.7), by fixing $\lambda_n = +\infty$, which means that $\forall i \notin \mathcal{I}_u, \boldsymbol{\alpha}_i = 1/m_\ominus$ (we call this sub-vector $\boldsymbol{\alpha}_\ominus$):

$$\begin{aligned}
\boldsymbol{\alpha}_u^* &= \underset{\boldsymbol{\alpha}\in\mathbf{A}_u}{\operatorname{argmin}} \ \|\boldsymbol{\alpha}_\oplus^\intercal \mathbf{X}_\oplus - \boldsymbol{\mu}_\ominus\|^2 + \lambda_p\|\boldsymbol{\alpha}_\oplus\|^2 \\
&= \underset{\boldsymbol{\alpha}\in\mathbf{A}_u}{\operatorname{argmin}} \ \|\boldsymbol{\alpha}_\oplus^\intercal \mathbf{X}_\oplus\|^2 - \|\boldsymbol{\mu}_\ominus\|^2 - 2\boldsymbol{\alpha}_\oplus^\intercal \mathbf{X}_\oplus^\intercal \boldsymbol{\mu}_\ominus + \lambda_p\|\boldsymbol{\alpha}_\oplus\|^2 \\
&= \underset{\boldsymbol{\alpha}\in\mathbf{A}_u}{\operatorname{argmin}} \ \boldsymbol{\alpha}_\oplus^\intercal \mathbf{X}_\oplus^\intercal \mathbf{X}_\oplus \boldsymbol{\alpha}_\oplus + \lambda_p\|\boldsymbol{\alpha}_\oplus\|^2 - 2\boldsymbol{\alpha}_\oplus^\intercal \mathbf{X}_\oplus^\intercal \boldsymbol{\mu}_\ominus,
\end{aligned} \tag{10.1}$$

where

$$\boldsymbol{\mu}_\ominus = \sum_{i\notin\mathcal{I}_u} \boldsymbol{\alpha}_i \mathbf{x}_i = \frac{1}{m_\ominus} \sum_{i\notin\mathcal{I}_u} \mathbf{x}_i = \boldsymbol{\alpha}_\ominus \mathbf{X}_\ominus$$

is the centroid of the convex hull spanned by the negative items and $\boldsymbol{\alpha}_\oplus$ are the weights associated with the positive items, $\mathbf{X}_\oplus$ and $\mathbf{X}_\oplus$ are the sub-matrices of $\mathbf{X}$ containing only the columns corresponding to the positive and the negative items, respectively. Note that all norms $\|\cdot\|$ in Equation 10.1 are intended as L^2-norm, that is $\|\cdot\|_2$, and, when not specified differently, the same abbreviation is assumed in the following. The number of parameters in (10.1) is equal to $m_\oplus$ and thus the complexity drops from $\mathcal{O}(n_{\text{TS}}m^2)$ to $\mathcal{O}(n_{\text{TS}}\overline{m}_\oplus{}^2)$, where $\overline{m}_\oplus = \mathbb{E}[|\mathcal{I}_u|]$ is the expected number of the positive items per user.

Implementation detail

Although the large improvement in terms of complexity, a naïve implementation could have an additional cost due to the calculation of $\boldsymbol{\mu}_{\ominus}$. Recalculating such value from scratch for each user would have a complexity of $\mathcal{O}(n_{\text{TS}} n \overline{m}_{\ominus})$, where $\overline{m}_{\ominus} = \mathbb{E}[|\mathcal{I} \setminus \mathcal{I}_u|]$, and it can be approximated with $O(n_{\text{TS}} nm)$.

To overcome this possible issue, we propose an efficient incremental way of calculating $\boldsymbol{\mu}_{\ominus}$. Let us consider the mean over all items

$$\boldsymbol{\mu} = \frac{1}{m} \sum_{i \in \mathcal{I}} \mathbf{x}_i,$$

then, for a given user u, we have

$$\boldsymbol{\mu}_{\ominus} = \frac{1}{m_{\ominus}} \left(m \cdot \boldsymbol{\mu} - \sum_{i \in \mathcal{I}_u} \mathbf{x}_i \right).$$

From a computational stand point, it is suffice to compute the sum $\sum_{i \in \mathcal{I}} \mathbf{x}_i$ only once (i.e., $m \cdot \boldsymbol{\mu}$) and then, for every $\boldsymbol{\mu}_{\ominus}$, subtract the sum of the positive items. Using this simple trick, the overall complexity drops to $\mathcal{O}(nm) + \mathcal{O}(n_{\text{TS}}^2 \overline{m}_{\oplus})$. In the remainder we will refer to this method as ECF-OMD.

10.2 Kernelized CF-OMD

The CF method proposed in Section 10.1, can be seen as a particular case of a kernel method. In fact, inside the optimization problem (10.1) we have the matrix $\mathbf{X}_{\oplus}^{\intercal} \mathbf{X}_{\oplus}$ that is actually a kernel matrix, in particular a linear kernel matrix. Let us call this matrix $\mathbf{K}_{\oplus}$ with the corresponding kernel function κ. Thus, we can reformulate (10.1) as:

$$\boldsymbol{\alpha}_u^* = \underset{\boldsymbol{\alpha}_{\oplus} \in \mathbf{A}_u}{\operatorname{argmin}} \ \boldsymbol{\alpha}_{\oplus}^{\intercal} \mathbf{K}_{\oplus} \boldsymbol{\alpha}_{\oplus} + \lambda_p \|\boldsymbol{\alpha}_{\oplus}\|^2 - 2 \boldsymbol{\alpha}_{\oplus}^{\intercal} \mathbf{q}, \tag{10.2}$$

where the elements of the vector $\mathbf{q} \in \mathbb{R}^{m_{\oplus}}$ are defined as

$$\mathbf{q}_i = \frac{1}{m_{\ominus}} \sum_{j \notin \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j). \tag{10.3}$$

Throughout the thesis we will refer to this method as CF-KOMD. It is worth to notice that, inside the optimization problem (10.2) any kernel function can be plugged.

The induced ranking is obtained using the scoring function

$$\hat{\mathbf{r}}_u = \mathbf{X}^\intercal \mathbf{w}_u^* = \mathbf{K}_{\oplus,:}^\intercal \boldsymbol{\alpha}_\oplus - \mathbf{q} \tag{10.4}$$

where $\mathbf{K}_{\oplus,:} \in \mathbb{R}^{|\mathcal{I}_u| \times |\mathcal{I}|}$ is the matrix which contains the subset of rows corresponding to the positive set of items for the user u.

A nice property of the optimization problem (10.2) is its insensibility to the addition of constant kernel matrices. In other words, adding a constant to the whole kernel matrix does not affect the solution of CF-KOMD (this can be demonstrated also for CF-OMD and KOMD).

Proof. Let $\mathbf{K} = \mathbf{K}_0 + \widehat{\mathbf{K}}$ be a kernel matrix where $\mathbf{K}_0 = k_0 \mathbf{1}_{m \times m}, k_0 > 0$, is a constant matrix. Let also $\widehat{\kappa}$ and κ_0 be the kernel functions associated to $\widehat{\mathbf{K}}$ and $\mathbf{K}_0$, respectively. Then, the vector $\mathbf{q}$ can be defined as:

$$\begin{aligned}
\mathbf{q}_i &= \frac{1}{m_\ominus} \sum_{j \notin \mathcal{I}_u} \left(\kappa_0(\mathbf{x}_i, \mathbf{x}_j) + \widehat{\kappa}(\mathbf{x}_i, \mathbf{x}_j) \right) \\
&= \frac{1}{m_u^-} \left[m_\ominus \cdot k_0 + \sum_{j \notin \mathcal{I}_u} \widehat{\kappa}(\mathbf{x}_i, \mathbf{x}_j) \right] \\
&= k_0 + \hat{\mathbf{q}}_i \Rightarrow \mathbf{q} = \mathbf{k}_0 + \hat{\mathbf{q}},
\end{aligned}$$

where $\mathbf{k}_0 = k_0 \mathbf{1}_n$. Now, we can rewrite the optimization problem (10.2) as (we omit the $\oplus$ subscription for brevity):

$$\begin{aligned}
\boldsymbol{\alpha}_u^* &= \underset{\boldsymbol{\alpha} \in \mathbf{A}_u}{\operatorname{argmin}} \;\; \boldsymbol{\alpha}^\intercal (\mathbf{k}_0 + \widehat{\mathbf{K}}) \boldsymbol{\alpha} + \lambda_p \|\boldsymbol{\alpha}\|^2 - 2 \boldsymbol{\alpha}^\intercal (\mathbf{k}_0 + \hat{\mathbf{q}}) \\
&= \underset{\boldsymbol{\alpha} \in \mathbf{A}_u}{\operatorname{argmin}} \;\; \boldsymbol{\alpha}^\intercal \mathbf{K}_0 \boldsymbol{\alpha} + \boldsymbol{\alpha}^\intercal \widehat{\mathbf{K}} \boldsymbol{\alpha} + \lambda_p \|\boldsymbol{\alpha}\|^2 - 2 \boldsymbol{\alpha}^\intercal \mathbf{k}_0 - 2 \boldsymbol{\alpha}^\intercal \hat{\mathbf{q}},
\end{aligned}$$

where both $\boldsymbol{\alpha}^\intercal \mathbf{K}_0 \boldsymbol{\alpha}$ and $-2 \boldsymbol{\alpha}^\intercal \mathbf{k}_0$ are constant values independent from $\boldsymbol{\alpha}$,

$$\begin{aligned}
\boldsymbol{\alpha}^\intercal \mathbf{K}_0 \boldsymbol{\alpha} &= k_0 \sum_{i \in \mathcal{I}_u} \sum_{j \in \mathcal{I}_u} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j = k_0 \sum_{i \in \mathcal{I}_u} \boldsymbol{\alpha}_i \sum_{j \in \mathcal{I}_u} \boldsymbol{\alpha}_j = k_0; \\
-2 \boldsymbol{\alpha}^\intercal \mathbf{k}_0 &= -2 \sum_{i \in \mathcal{I}_u} k_0 \boldsymbol{\alpha}_i = -2 k_0 \sum_{i \in \mathcal{I}_u} \boldsymbol{\alpha}_i = -2 k_0;
\end{aligned}$$

and thus the solution of the optimization problem does not depend on the constant matrix $\mathbf{K}_0$:

$$\boldsymbol{\alpha}_u^* = \underset{\boldsymbol{\alpha} \in \mathbf{A}_u}{\operatorname{argmin}} \;\; \boldsymbol{\alpha}^\intercal \widehat{\mathbf{K}} \boldsymbol{\alpha} + \lambda_p \|\boldsymbol{\alpha}\|^2 - 2 \boldsymbol{\alpha}^\intercal \hat{\mathbf{q}},$$

and this is the same as the optimization problem (10.2). □

Approximation of $\mathbf{q}$

It is possible to further lower the complexity by providing a good approximation of $\mathbf{q}_u$ that can be computed only once, instead of n_{ts} times. The idea consists in replacing every $\mathbf{q}_{ui}$ with an estimate of $\mathbb{E}[\kappa(\mathbf{x}_i, \mathbf{x})]$ which is the expected value of the kernel between the item i and every other items. Formally, let us consider, without any loss of generality, a normalized kernel function κ and let the approximation of $\mathbf{q}$ be $\hat{\mathbf{q}}$ such that:

$$\hat{\mathbf{q}}_i = \frac{1}{m} \sum_{j \in \mathcal{I}} \kappa(\mathbf{x}_i, \mathbf{x}_j), \quad \forall\, i. \tag{10.5}$$

At each component of $\hat{\mathbf{q}}$, the approximation error is bounded by $2m_{\oplus}m^{-1}$, which is linear on the sparsity of the dataset.

Proof.

$$\begin{aligned}
|\hat{\mathbf{q}}_i - \mathbf{q}_i| &= \left| \frac{1}{m} \sum_{j \in \mathcal{I}} \kappa(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{m_{\ominus}} \sum_{j \notin \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) \right| \\
&= \left| \frac{1}{m} \left[\sum_{j \in \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{j \notin \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) \right] - \frac{1}{m_{\ominus}} \sum_{j \notin \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) \right| \\
&= \left| \frac{1}{m} \sum_{j \in \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) - \frac{m - m_{\ominus}}{m \cdot m_{\ominus}} \sum_{j \notin \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) \right| \\
&\leq \left| \frac{1}{m} \sum_{j \in \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) \right| + \left| \frac{m - m_{\ominus}}{m \cdot m_{\ominus}} \sum_{j \notin \mathcal{I}_u} \kappa(\mathbf{x}_i, \mathbf{x}_j) \right| \\
&\leq \left| \frac{m_{\oplus}}{m} \right| + \left| \frac{m - m_{\ominus}}{m \cdot m_{\ominus}} m_{\ominus} \right| \leq \frac{m_{\oplus} + m - m_{\ominus}}{m} = 2\frac{m_{\oplus}}{m}.
\end{aligned}$$

□

10.3 Sparsity and long tail distribution

As mentioned in Section 10.1, CF datasets are, in most of the cases, very sparse and in general the distribution of the ratings assume a long tail form [And06]. From the items perspective, this means that a small set of items, the most popular ones, receive great part of the whole set of ratings.

In order to study the behaviour of the kernel functions on CF datasets, we would like to understand which are the conditions under which kernel matrices are sparse and in which ones, instead, they tend to be dense. For studying this phenomenon we use the linear kernel as a representative example, but results also apply for every dot-product kernel as well because they can be "sparsified" in such a way that their zero entries are the same as for the linear kernel.

This sparsification procedure is based on a well known result from harmonic theory [KK12] that states:

Theorem 10.1

A function $f : \mathbb{R} \to \mathbb{R}$ defines a positive definite kernel $\kappa : \mathbf{B}(0,1) \times \mathbf{B}(0,1)$ as $\kappa : (\mathbf{x}, \mathbf{z}) \mapsto f(\langle \mathbf{x}, \mathbf{z} \rangle)$ iff f is an analytic function admitting a Maclaurin expansion with non-negative coefficients, $f(x) = \sum_{s=0}^{\infty} a_s x^s, a_s \geq 0$.

As emphasized in [KK12, DA16], many kernels used in practice [SS01] satisfy the above-mentioned condition.

From Theorem 10.3 we can observe that the kernel matrices induced by these kernels defined by that expansion are, in general, dense due to the zero degree term (i.e., $s = 0$) which is a constant added to all the entries of the matrix. Since we have demonstrated that adding a constant matrix to the kernel does not change the solution of CF-KOMD, we can "sparsify" these kernels by removing the zero degree factor, obtaining kernel matrices whose sparsity depends only on the distribution of the input data. This also implies that the sparsity of these sparsified kernels are exactly the same as of the linear kernel.

So, let $\mathbf{K} = \mathbf{X}^\intercal \mathbf{X}$ ($\mathbf{X} \in \mathbb{R}^{n \times m}$) be a kernel matrix and let $\mathbb{P}(\mathbf{K}_{i,j} \neq 0)$ be the probability that an entry $\mathbf{K}_{i,j}$ is non-zero. Given an a-priori probability distribution over the ratings, and assuming the independence of the ratings, we can estimate the probability of having a non-zero value in the Gram matrix with:

$$
\begin{aligned}
\mathbb{P}\left(\mathbf{K}_{i,j} \neq 0\right) &= 1 - \mathbb{P}\left(\mathbf{K}_{i,j} = 0\right) \\
&= 1 - \prod_h \mathbb{P}\left(\mathbf{X}_{i,h}\mathbf{X}_{j,h} = 0\right) \\
&= 1 - \prod_h (1 - \mathbb{P}\left(\mathbf{X}_{i,h}\mathbf{X}_{j,h} \neq 0\right)) \\
&= 1 - \prod_h (1 - \mathbb{P}\left(\mathbf{X}_{i,h} \neq 0\right)\mathbb{P}\left(\mathbf{X}_{j,h} \neq 0\right)) \\
&= 1 - (1 - \mathbb{P}\left(\mathbf{X}_{i,h} \neq 0\right)\mathbb{P}\left(\mathbf{X}_{j,h} \neq 0\right))^n
\end{aligned}
\tag{10.6}
$$

where $\mathbb{P}\left(\mathbf{X}_{i,h} \neq 0\right)$ and $\mathbb{P}\left(\mathbf{X}_{j,h} \neq 0\right)$ are the probability of having a non-zero entry in

the rating matrix.

It is worth to notice that, assuming the uniform distribution, such probability is actually the density of the matrix. However, it does not take into account the fact that all the elements in the diagonal of the kernel are for sure non zero (assuming non null examples). Hence, we can estimate the kernel density $d(\mathbf{K})$, with $\mathbf{K} \in \mathbb{R}^{m \times m}$, as in the following:

$$d(\mathbf{K}) = \frac{1}{m^2}\left[m + (m^2 - m)\mathbb{P}(\mathbf{K}_{i,j} \neq 0)\right]. \tag{10.7}$$

By looking at Equation (10.7) and (10.6), we can argue that anytime both $\mathbf{X}_i$ and $\mathbf{X}_j$ are popular items, and hence, $\mathbb{P}(\mathbf{X}_{i,h} \neq 0)$ and $\mathbb{P}(\mathbf{X}_{j,h} \neq 0)$ are close to 1, then $\mathbb{P}(\mathbf{K}_{i,j} \neq 0)$ tends to be high and $\mathbf{K}$ is likely to be dense. On the other hand, when one of the vectors represents an unpopular item, then the probability $\mathbb{P}(\mathbf{K}_{i,j} \neq 0)$ is likely close to zero. Considering that we are assuming a long tail distribution over the items, most of the kernel entries which correspond to dot-product of two unpopular have few users that rated both of them and so such kernel entries tend to be sparse.

Though, all our considerations are based on the assumption that users are uniformly distributed and in real datasets this is not often the case.

If we assume a long tail distribution over the users, the probability of having at least one user in common between two items would be generally high, since the ratings for an item are likely to be concentrated on the (few) most active users.

10.3.1 Empirical analysis of CF datasets

In order to validate our previous considerations, we empirically analyzed a set of CF datasets comparing the theoretical sparsity with uniform ratings distribution with the actual sparsity of the linear kernel.

The empirical analysis have been performed as follows. For each dataset, we built the corresponding rating matrix $\mathbf{R}$, we calculated the expected density using (10.6) by fixing $\mathbb{P}(\mathbf{X}_{i,h} \neq 0)$ equals to the density of $\mathbf{R}$. Then, we computed the linear kernel $\mathbf{K}_{\text{LIN}} = \mathbf{R}^\intercal\mathbf{R}$ and finally we compared the theoretical sparsity $d(\mathbf{K}_{\text{LIN}})$. Table 10.1 summarizes the empirical results. Even though our intuitions seem to be confirmed by most of the datasets, we can notice that for `BookCrossing` and `Ciao` $\mathbf{K}_{\text{LIN}}$ is more dense than the estimate. To further analysis this strange behaviour for each dataset we plotted its distribution over both users and items. These plots are depicted in Figure 10.1 and Figure 10.2. The plots are in *loglog* scale and the blue line represents the best fitting power low function. The fitting has been made using the least square method.

From the plots we can observe that:

[6] For this analysis we used the 10M version of the MovieLens dataset.

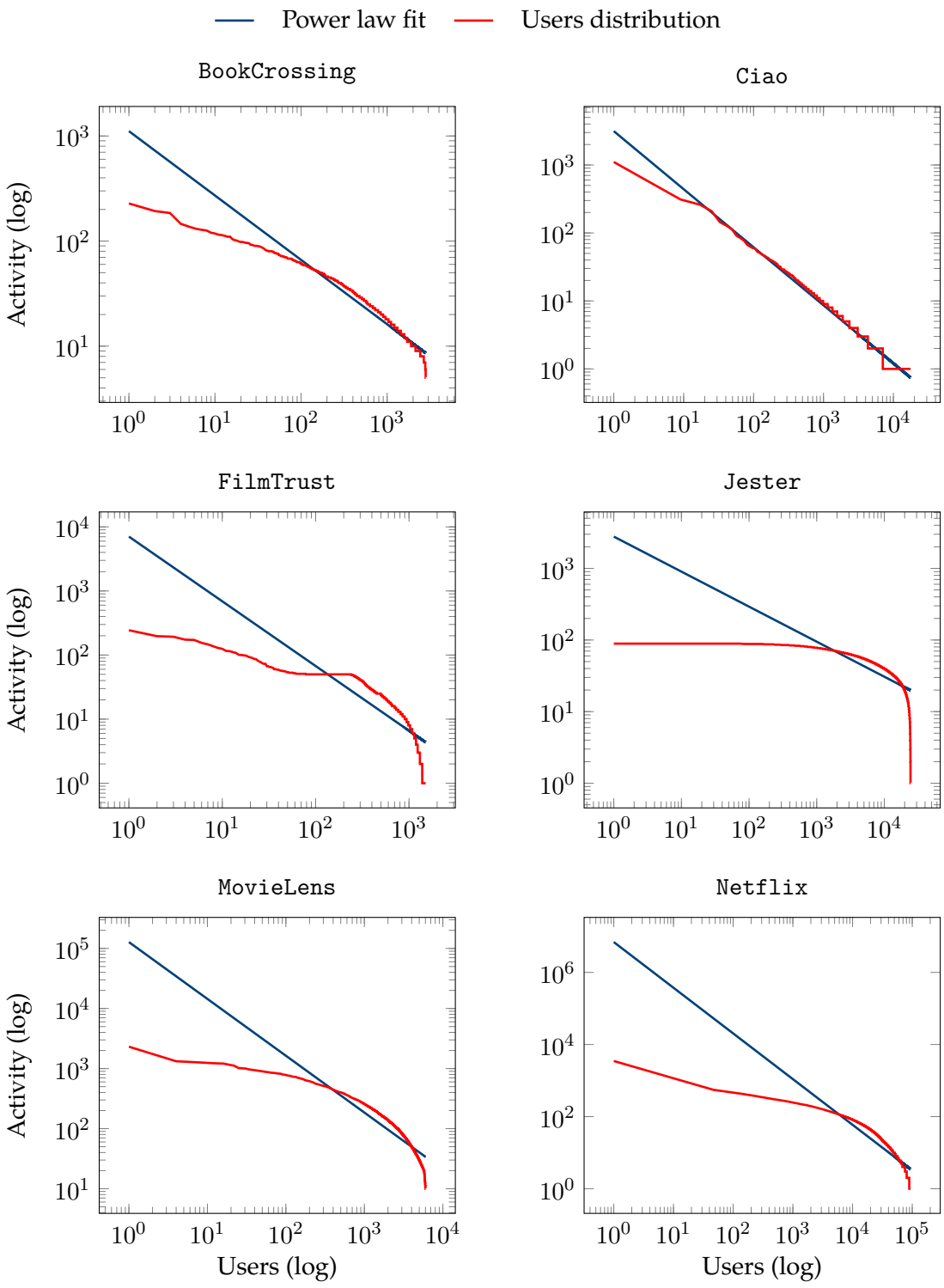


Figure 10.1: Users' distribution of the datasets. In blue is depicted the best fitting power law curve. The plot is in log scale.

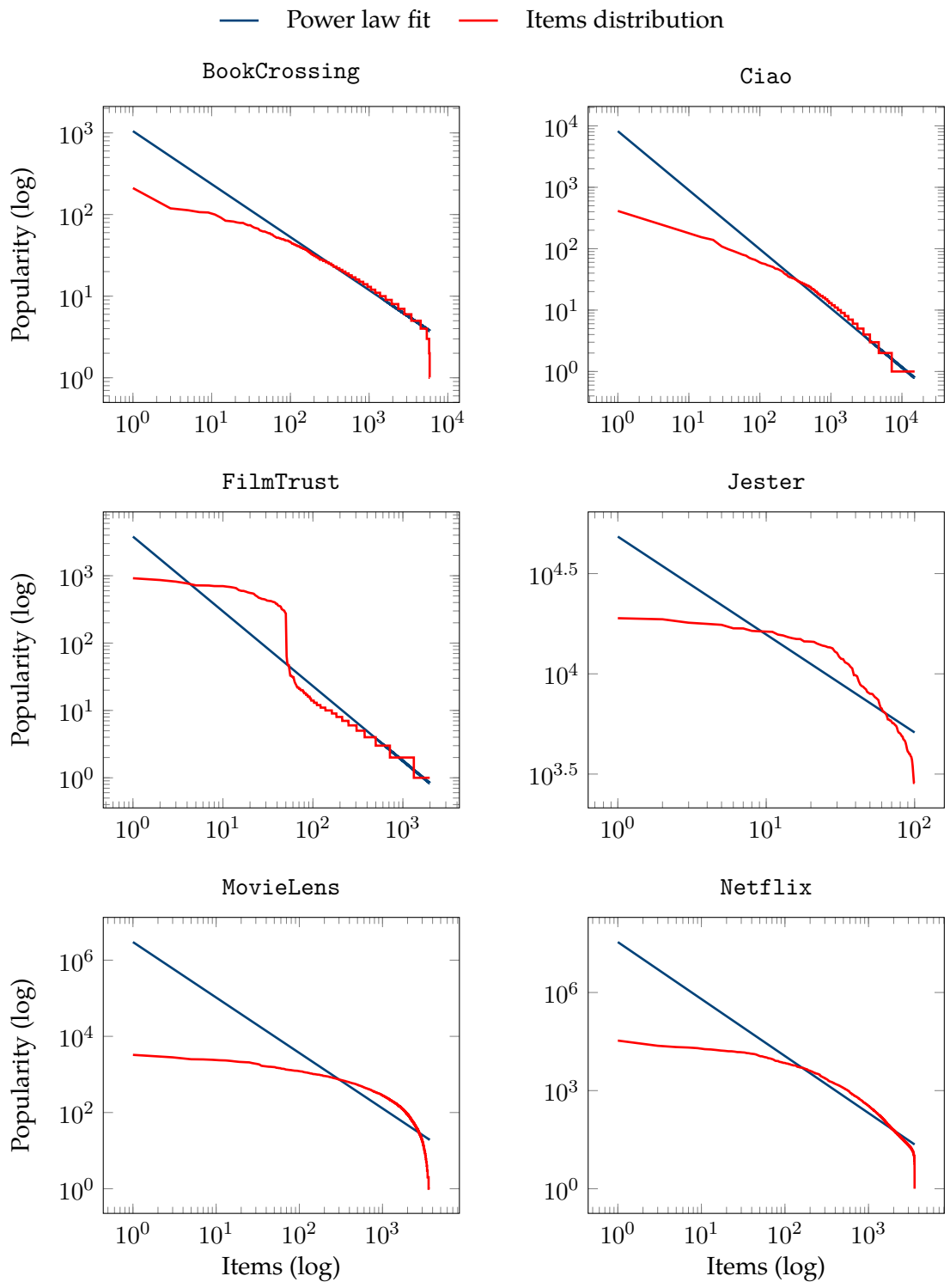


Figure 10.2: Items' distribution of the datasets. In blue is depicted the best fitting power law curve. The plot is in log scale.

Dataset	Density R	Density $\mathbf{K}_{\text{LIN}}$	$d(\mathbf{K}_{\text{LIN}})$
BookCrossing	0.003%	0.687%	0.011%
Ciao	0.025%	2.51%	0.12%
FilmTrust	1.13%	11.11%	17.74%
Movielens [6]	1.34%	85.56%	99.99%
Netflix	0.99%	98.03%	99.99%

Table 10.1: Analysis of the sparsity of the linear kernel.

- none of the items' distributions follows exactly a power low, especially the head of the distribution;
- datasets with a very dense linear kernel tend to have shorter tails;
- generally items are long tailed while users tend to be more uniform distributed;
- users distributions are in general not well fitted by a power law, with the exception of Ciao and BookCrossing datasets.

IT is clear that Ciao and BookCrossing are the only two datasets with a well defined long tail distribution over both the users and the items. This confirm our intuition about the likelihood of having a more dense kernel with both the long tailed distributions. This represents a crucial point of our analysis: having a long tailed users' distribution belong with the long tailed items' distribution, increases the chances of having a non-zero dot-product $\mathbf{X}_i^\intercal \mathbf{X}_j$ because, even though the items are sparse, the non-zero values are concentrated on a small fraction of the users.

We can conclude that, the long tail distribution over the items keeps the kernel sparse while a long tail distribution over the users increases its density.

10.4 Evaluation

In this section we evaluate our proposals on different aspects: scalability, performance, and time complexity.

10.4.1 Scalability

Before going deep in the performance evaluation of the proposed kernel-based method against several state-of-the-art algorithm, in this section we want to test the scalability of our proposal on a very large scale dataset. Specifically, we used the `MSD` dataset to compare the training time and the performance of ECF-OMD and CF-KOMD against the winner of the MSD Kaggle challenge, i.e., MSDw (see Section 5.1). We used the same setting as the challenge, that is: the training set is composed by 1M users (plus 10K users as validation set) with all their listening history and for the rest (i.e., 100K users) only the first half of the history is provided, while the other half constitutes the test set.

In these experiments we fixed the ECF-OMD and CF-KOMD hyper-parameter λ_p to 0.01 (different values do not change the solution so much). Cf-KOMD has been tested with the Homogeneous polynomial kernel of degree 2, while the setting of the MSDw algorithm is the same used to win the challenge [Aio13].

The achieved results (AUC and map@500) are presented in Table 10.2. As we can notice, MSDw maintains its leadership performance in terms of mAP@500 (the same metric used in the challenge), while for the AUC all the methods have very similar performance. This underline the fact that both ECF-OMD and CF-KOMD are designed to optimize the AUC rather than the mAP.

Metric	MSDw	ECF-OMD	CF-KOMD
mAP@500	**0.169**	0.164	0.160
AUC	**0.973**	0.970	0.970

Table 10.2: Ranking accuracy on `MSD` using AUC and mAP@500. In **bold** are highlighted the best results.

The computational costs of all the compared methods on this dataset are reported in Figure 10.3.

The computational times reported in Figure 10.3 are the average over one thousand test users. All methods run on a machine with 150Gb of RAM and 2×Eight-Core Intel ® Xeon ® CPU E5-2680 0 @ 2.70GHz. Note that the times in the figure have a constant overhead due to read operations. Results show that ECF-OMD and CF-KOMD are al-

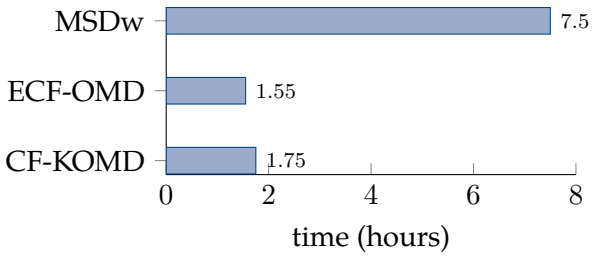


Figure 10.3: Average computational time in hours for 1K users.

most 5 time faster than MSDw, however, they require more RAM to store the kernel matrices. It is worth to notice that CF-KOMD has a computational time very close to ECF-OMD, and this highlights the positive effects of the complexity optimization presented in Section 10.2.

10.4.2 Top-n recommendation

Evaluation protocol

We compared the CF methods in terms of AUC (see Chapter 3) and for each dataset we performed 5 repetition of the test with different splits (the same for all methods). In particular, each dataset has been pre-processed as in the following:

1. we split randomly the users in 5 sets of the same dimension;
2. for each user in a set we further split its ratings in two halves;
3. at each round test, we use all the ratings in 4 sets of users plus the first half of ratings of the remaining set as training set, and the rest as test set.

This setting avoids situations of cold start for users, because in the training phase we have at least some ratings for every user. The results reported in the figures are the average AUCs, with their standard deviation, over the 5 folds. In these experiments, for computational reasons, we did not consider the `MSD` dataset.

Compared methods

In the following we list all the methods and the parameters tested in our experiments:

MSDw [Aio13] the winner method of the MSD challenge described in Chapter 5. The asymC parameter α has been tested in the set $\{0, 0.15, 0.25, 0.5, 0.75, 1\}$;

ECF-OMD the efficient version of CF-OMD. For this algorithm we fix $\lambda = 0.1$ since other values do not change the solution very much;

CF-KOMD the kernelized version of ECF-OMD. In particular we tested the following kernels:

Tanimoto (CF-K_T) which does not have parameters;

Polynomial (CF-K_{HP}) homogeneous of degree 2;

mC-kernel (CF-K_{mC}) with different degrees greater or equal than 2.

mD-kernel (CF-K_{mD}) with different degrees greater or equal than 2.

Non-monotone kernels are not adapt for this kind of application since the negative data points are actually unknown (i.e., unlabelled).

SLIM [NK11] described in Chapter 5. We tested β in the range [1,5] and λ in the set $\{0.1, 0.5, 1, 2\}$;

WRMF [HKV08] described in Chapter 5. Since we use implicit feedback we fix the value of α to 1. We also fixed the number of factors to 50 and the learning rate to 0.01, while the regularization parameter λ has been tested in the range $\{1, 10, 100, 300\}$;

BPR-MF [RFGST09] described in Chapter 5. Also for this algorithm we fixed the number of factors to 50 and the learning rate to 0.01, while the regularization parameters have been fixed to 0.0025.

We implemented all methods, but SLIM, in python and the source code is freely available on *github* at `https://github.com/makgyver/pyros`. Regarding SLIM, we used the implementation provided by the authors, that can be downloaded at `https://www-users.cs.umn.edu/~ningx005/slim/html/`.

Experimental results

The best performance of the tested algorithms are reported in Table 10.3. It is worth to notice that we did not perform validation but, for each method, we selected the best average result on the test set over all the hyper-parameters configuration. Since we exhaustively try he configuration indicated in the previous section, the procedure is quite fair, or at most might foster methods with many hyper-parameters.

In 4 out of 6 the best performing method is CF-KOMD with the mD-kernel as kernel function. In particular, the increasing of the AUC with respect to the linear case (i.e., ECF-OMD) is impressive, especially on `Ciao` and `Jester`. It is also worth to notice that, for the mD-kernel, the best score on `Ciao` is reached with degrees greater than 100 which are surprisingly high. However, this can be due to the combination of some peculiar features of this dataset, such as, its high degree of sparseness, its high number of features (i.e., users) and its double long tailed distribution. These characteristics entail a very slow decrease of the complexity with the increasing of d, as underlined in Figure 10.8. In general, `Ciao` has been a problematic dataset for most of the tested methods, the only exceptions are BPR-MF, MSDw and CF-KOMD with the disjunctive kernel.

[7] Using the implementation provided by the authors we were not able to do the test on this dataset.

Method	BookX	Ciao	FilmTrust	Jester	Movielens	Netflix
MSDw	0.743 ±0.004	0.824 ±0.008	0.961 ±0.004	0.727 ±0.001	0.875 ±0.001	0.939 ±0.0002
SLIM	0.714 ±0.004	0.788 ±0.016	0.969 ±0.004	0.658 ±0.003	0.861 ±0.005	-[7]
WRMF	0.739 ±0.004	0.755 ±0.007	0.956 ±0.005	**0.739** ±0.004	0.877 ±0.006	0.927 ±0.001
BPR-MF	0.691 ±0.005	**0.848** ±0.004	0.959 ±0.007	0.649 ±0.002	0.874 ±0.005	0.848 ±0.001
ECF-OMD	0.731 ±0.003	0.718 ±0.006	0.961 ±0.005	0.602 ±0.003	**0.896** ±0.0005	0.943 ±0.001
CF-K_{HP}	0.748 ±0.002	0.731 ±0.003	0.961 ±0.006	0.601 ±0.003	0.893 ±0.0003	0.937 ±0.001
CF-K_{T}	0.744 ±0.003	0.734 ±0.004	0.964 ±0.005	0.626 ±0.003	0.895 ±0.0004	0.941 ±0.001
CF-K_{mC}	0.695 ±0.005	0.540 ±0.009	0.951 ±0.008	0.601 ±0.003	0.892 ±0.0004	0.936 ±0.001
CF-K_{mD}	**0.751** ±0.005	0.841 ±0.006	**0.971** ±0.004	0.727 ±0.002	0.895 ±0.0004	**0.944** ±0.001

Table 10.3: AUC results on 6 CF datasets. Results are reported with their standard deviation. For each dataset the highest AUC is highlighted in **bold**.

Regarding the mC-kernel, results confirm that too high expressiveness do not lead to good performances. In fact, in all the cases its best results have been reached with $d = 2$. It is important to underline that the null vector problem described in Section 6.3.2 has artificially been solved by forcing in the kernel matrix a 1 in correspondence of the diagonal of these degenerate examples. Even though this is a reasonable fix, it is an arbitrary choice. Overall, the compared baselines have achieved good performance but in general a bit worse than our proposals bar WRMF on the `Jester` dataset, where CF-KOMD had very poor results with the exception of the mD-kernel, BPR on `Ciao` and SLIM on `FilmTrust`. In all these cases the differences are not statistically significant.

The problem with the compared approaches (excluding MSDw) is their huge number of hyper-parameters that need to be tuned, while CF-KOMD has only the hyper-parameters of the used kernel function, since the λ_p can be actually fixed to 0.01.

10.4.3 Computational complexity

In Section 6.2 we have shown that the computational complexity of the mD-kernel is $\mathcal{O}(n + d)$, where n is the number of dimension of the input vectors, and d the degree of the kernel. This complexity concerns only the computation of the kernel function between two binary vectors. In the calculation of the entire kernel matrix the number of examples (in this case the number of items) plays a huge role. Indeed, Figure 10.4 (left plot) shows that all the kernel matrix computations, but `Ciao`, is really fast.

On the other hand, by using the recursive method presented in Section the kernel computation can be (for low degrees) even faster. However, in this case, the time complexity increases linearly with the arity of the kernel because of the increasing of the number of recursive calls (see the right plot of Figure 10.4). Nonetheless, for low degrees the recursive method is more efficient than the standard one since it is based on matrix operations rather than entry by entry operations.

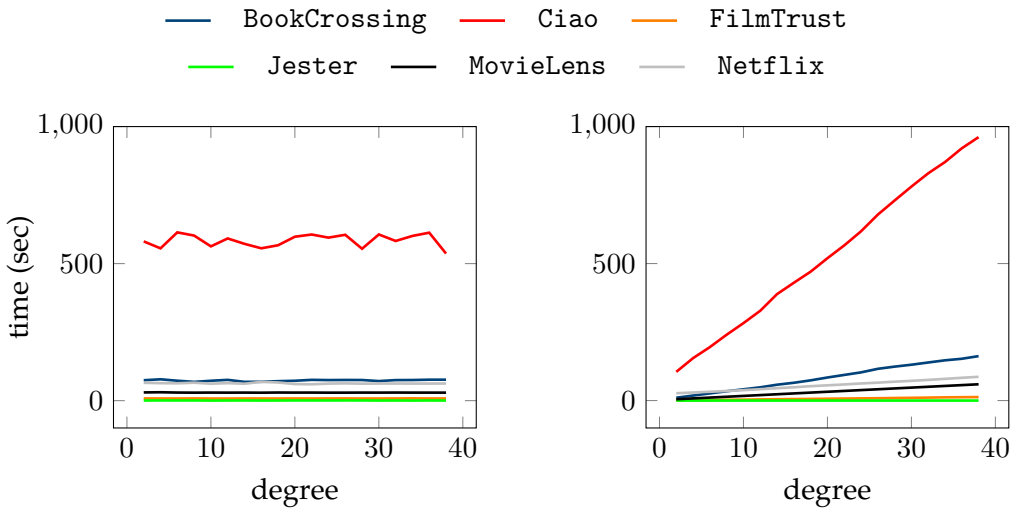


Figure 10.4: Time, in seconds, for calculating the mD-kernel on different datasets.

All these experiments have been performed on a MacBook Pro late 2012 with 16GB of RAM and CPU Intel ® Core™ i7 @ 2.70GHz.

10.4.4 Spectral ratio

In Section 6.2 we have proved how the spectral ratio is linked with the arity of the mC-kernel and the mD-kernel. To further support our theoretical proves, Figures 10.5 and Figure 10.6 show, over all the CF datasets, the behaviour of the mC-kernel and the mD-kernel SR varying the degree. We include in the plots the kernels of degree 1 (i.e., the linear kernel) for two reasons:

1. it gives a visual idea of how fast/slow the SR of the kernel matrix increases/decreases with respect to the linear one;
2. for each dataset, it offers a meeting point between the two curves (the mC-kernel and the mD-kernel curves).

Besides supporting our previous results, the plots show that augmenting the degree, the decrease of the SR in the mD-kernel is smoother than its increase in the mC-kernel, which reaches very rapidly values near to 1 (so the kernel matrix is vey similar to the identity kernel). This justifies the fast decay of the performance of the mC-kernel on all datasets.

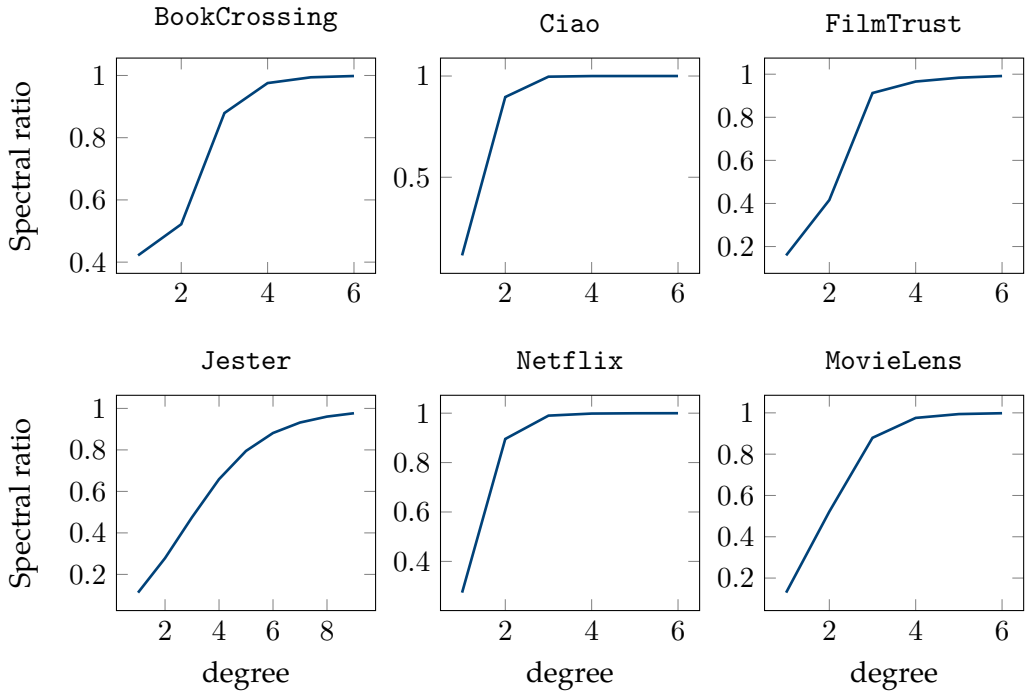


Figure 10.5: Spectral complexity of the mC-kernel varying the degree.

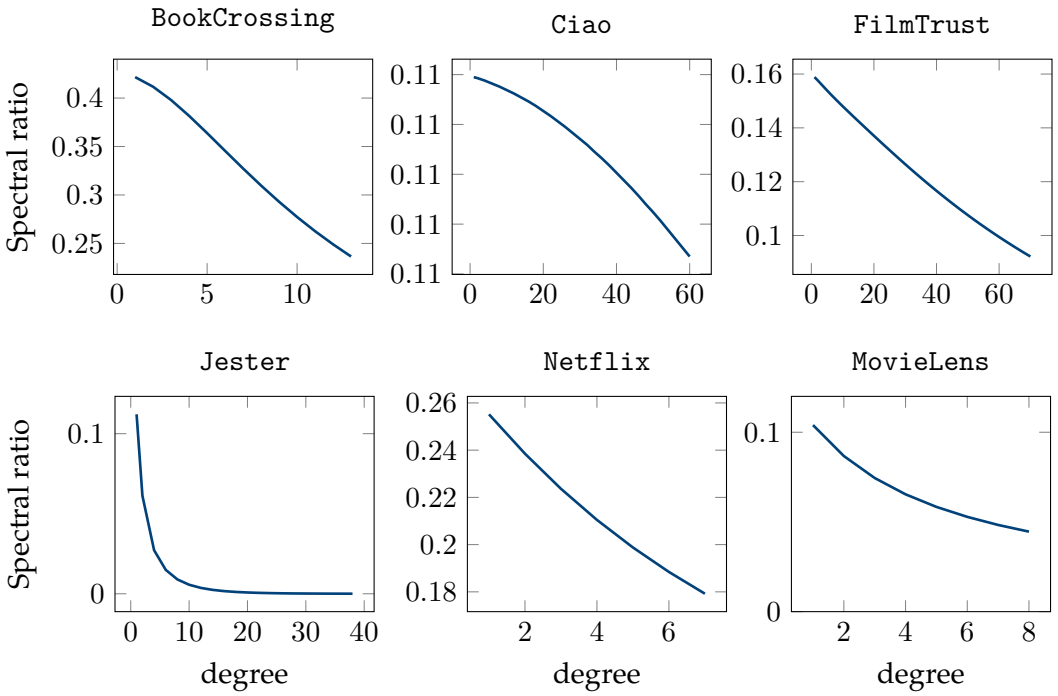


Figure 10.6: Spectral complexity of the mD-kernel varying the degree.

10.4.5 Effect of the degree on the Boolean kernels

In Figure 10.7 and Figure 10.8 are depicted the AUC scores achieved with different degrees of the mc-kernel and the mD-kernel, respectively. In both cases the trend of the plots are quite consistent: the performance of the mD-kernel slowly increase until it reaches a peak and then it starts to constantly decrease. In some cases the peak is actually in the smallest non-trivial degree $d = 2$ and thus the plot is always decreasing.

On the other hand, the performance of the mC-kernel rapidly decrease. This is a further confirmation that in these kind of datasets CF-KOMD works better by using kernels with low expressiveness, even lower than the linear kernel.

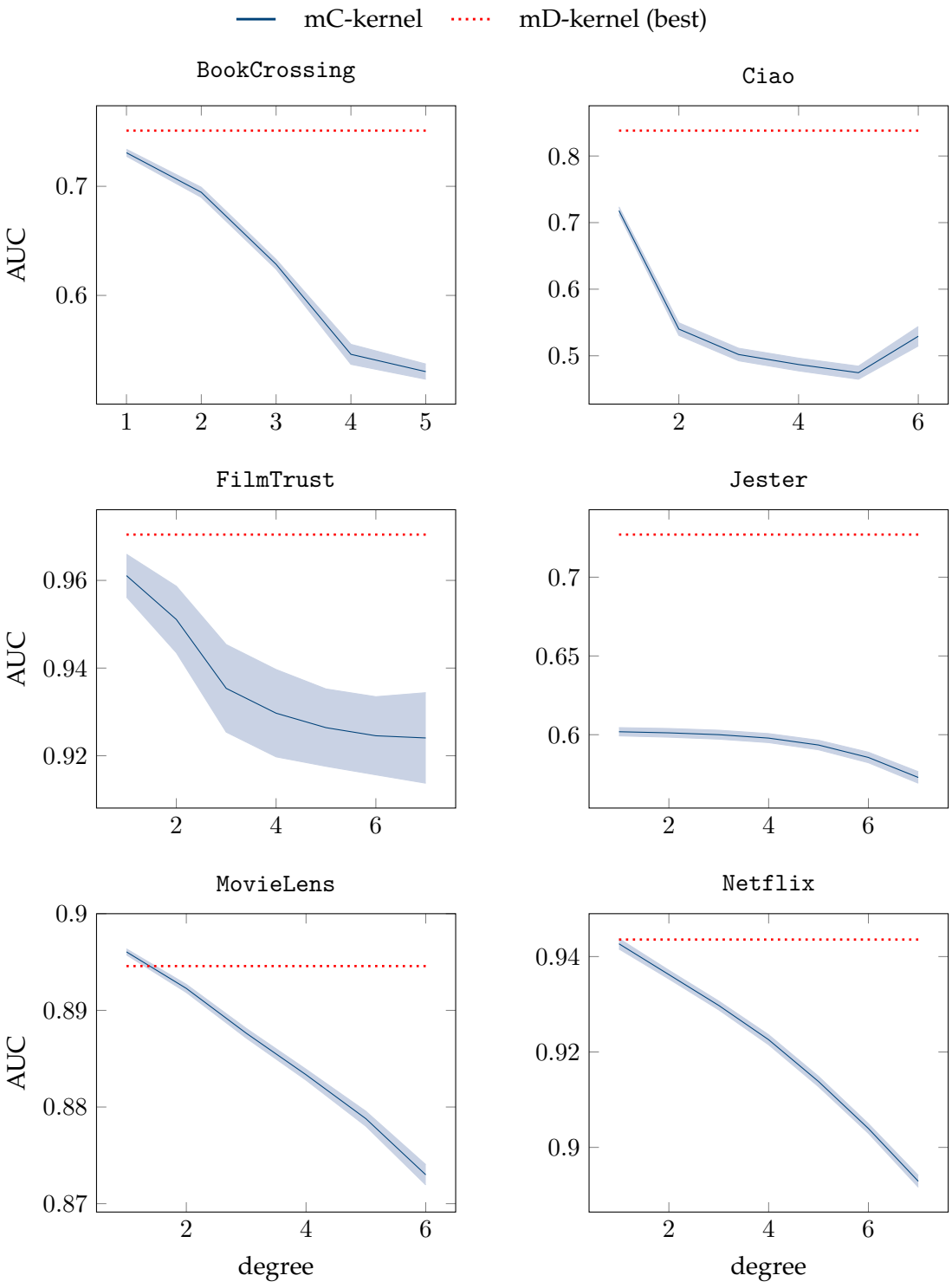


Figure 10.7: Performance of different mC-kernel degrees.

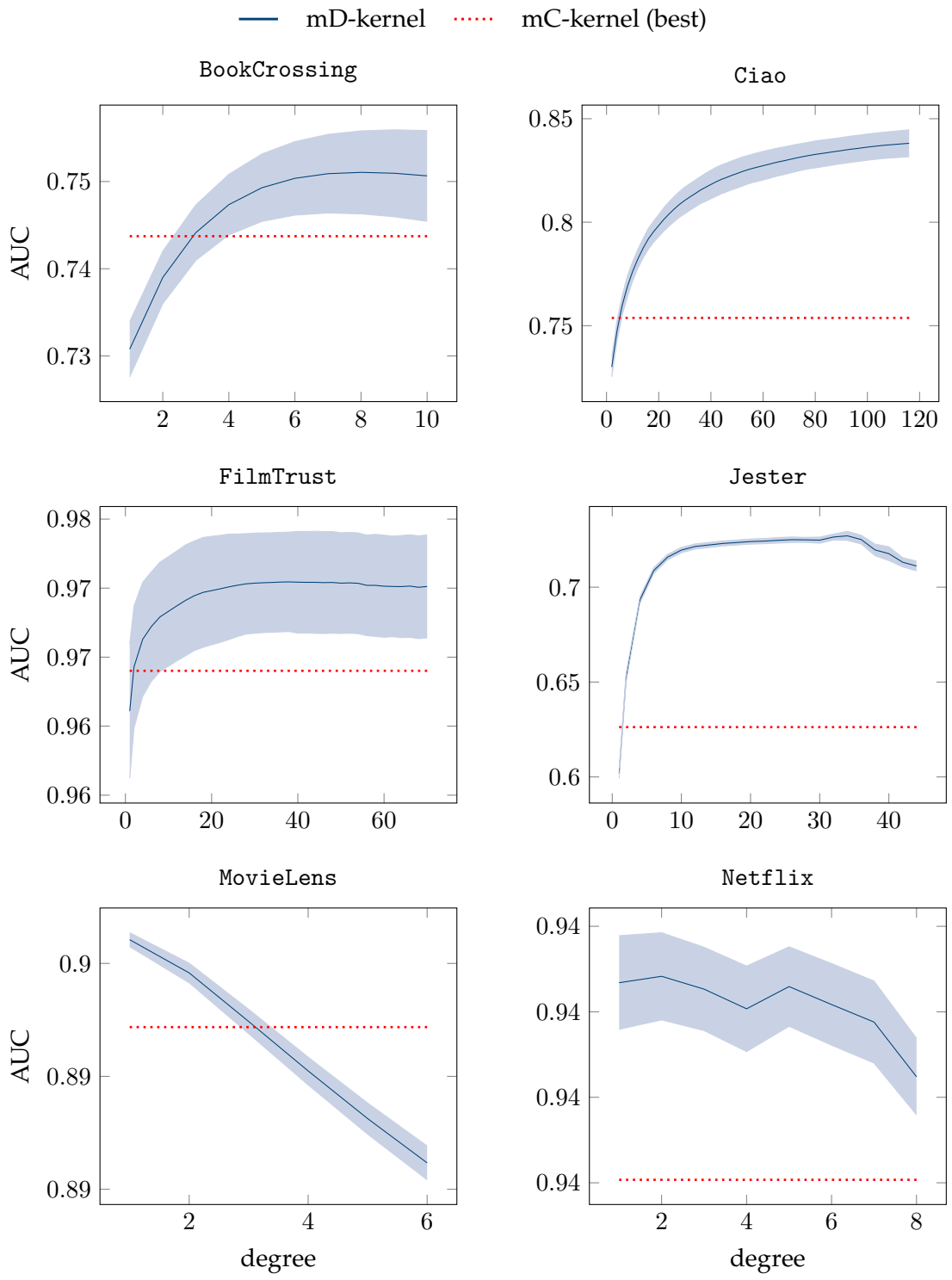


Figure 10.8: Performance of different mD-kernel degrees.

PART

IV Conclusions

Summary

11

Conclusions and future work

Do not go gentle into that good night,
Old age should burn and rave at close of day;
Rage, rage against the dying of the light.

Do not go gentle into that good night
Dylan Thomas

11.1 Conclusions

In this thesis we focused on the representation problem in machine learning by facing it from different perspectives. The first aim was to propose theoretical well founded tools that can ease the understanding of established machine learning models such as kernel methods. Besides the interpretability, we also wanted to provide methods which are able to achieve state-of-the-art performance on specific domains concerning categorical data.

The main contribution of the thesis is a new perspective on kernel functions. We proposed families of kernels with the goal of creating embedding spaces that can be understood and, consequently, that can be used to extract rules able to explain the decision of a kernel machine.

The first family of kernels, called Boolean kernels, offers the possibility of creating feature spaces composed of well formed logical formulas which allow to extract easy-to-read rules from the solution of a SVM. An example of such procedure is presented as well as an extensive evaluation of the kernels on several binary classification tasks.

The second proposed family of kernels, dubbed propositional kernels, try to overcome the limitations of the Boolean kernels, especially from the expressivity point of view. Propositional kernels are able to construct almost any logical proposition, and hence they can be useful when there is the need of more flexibility. We provided an elegant and efficient way to compute and generate these kernels as well as an example

of application.

On the other hand, we have also investigated in new algorithms for learning the best representation for a given task. Our second main contribution is in fact a multiple kernel learning algorithm, which is based on an optimization criterion that gives many theoretical guarantees about the quality of the solution. Extensive empirical evaluations showed that our proposal outperforms state-of-the-art MKL algorithms on several benchmark datasets.

Finally, the Boolean kernel framework has been applied in a very different domain w.r.t. the classification problem, namely on top-N recommendation tasks. Firstly, we have proposed a very efficient kernel-based collaborative filtering method. Then we applied on top of it our Boolean kernels on several recommender system datasets showing state-of-the-art results. A very important observation is the ability of the Boolean kernels, in particular the mD-kernel, of alleviating the sparsity issue typical in RSs.

11.2 Future work

By analyzing the weaknesses and the possible extensions of all the novelties of this thesis we can underline these possible paths that can be followed in the future:

- extend the applicability of our Boolean kernels to real-valued datasets. A possibility could be to study methods for binarizing real values in some smart way in order to loose as less information as possible. Heuristics, for example based on the entropy could be a first possible approach that deserves a try;
- application of the algorithm GRAM on datasets with real valued data, as well as, an in depth theoretical analysis about its convergence;
- development of efficient and theoretical well founded methods for interpreting the solution of kernel machines. Our proof of concept is only a first step in this direction, we aim to develop smarter and more efficient algorithms based, for example, on some heuristics. The next step will be to test such methods on real world datasets in order to see whether the extracted rules are easy to interpret also for non expert users;
- developing new methods for generating and extracting the best propositional kernel using the proposed framework;
- Studying whether it is possible to extend the propositional kernel framework to fuzzy logic.

Bibliography

[AD91] Hussein Almuallim and Thomas G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, AAAI'91, pages 547–552. AAAI Press, 1991. [Cited on pages 85 and 86.]

[AD15] Fabio Aiolli and Michele Donini. Easymkl: a scalable multiple kernel learning algorithm. *Neurocomputing*, pages 215–224, 2015. [Cited on pages 5, 43, and 115.]

[ADNS15] F. Aiolli, M. Donini, N. Navarin, and A. Sperduti. Multiple graph-kernel learning. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 1607–1614, 2015. [Cited on pages 5, 41, and 42.]

[ADSMS08] Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. *A Kernel Method for the Optimization of the Margin Distribution*, pages 305–314. 2008. [Cited on pages 33 and 51.]

[Aio05] Fabio Aiolli. A preference model for structured supervised learning tasks. *IEEE International Conference on Data Mining*, pages 557–560, 2005. [Cited on page 51.]

[Aio13] Fabio Aiolli. Efficient top-N recommendation for very large scale binary rated datasets. In *ACM Recommender Systems Conference*, pages 273–280, Hong Kong, China, 2013. [Cited on pages 48, 50, 137, and 138.]

[Aio14] Fabio Aiolli. Convex AUC optimization for top-N recommendation with implicit feedback. In *ACM Recommender Systems Conference*, pages 293–296, New York, USA, 2014. [Cited on pages 48, 51, and 128.]

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. [Cited on page 3.]

[And06] Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006. [Cited on page 131.]

[BA12] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer Publishing Company, Incorporated, 3rd edition, 2012. [Cited on page 96.]

[BAJB17] Andrey Bondarenko, Ludmila Aleksejeva, Vilen Jumutc, and Arkady Borisov. Classification tree extraction from trained artificial neural networks. *Procedia Computer Science*, 104(Supplement C):556 – 563, 2017. [Cited on page 4.]

[BB10] Nahla Barakat and Andrew P. Bradley. Rule extraction from support vector machines: A review. *Neurocomputing*, 74(1-3):178–190, December 2010. [Cited on pages 5 and 89.]

[BCS14] Guy Bresler, George H. Chen, and Devavrat Shah. A latent source model for online collaborative filtering. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3347–3355, Cambridge, MA, USA, 2014. MIT Press. [Cited on page 50.]

[BCS+16] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4945–4949, 2016. [Cited on page 4.]

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. [Cited on page 4.]

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. [Cited on pages 28 and 34.]

[BHN05] Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(1):38–46, January 2005. [Cited on page 43.]

[BL] J. Bennet and S. Lanning. The netflix prize. In *KDD Cup and Workshop*. [Cited on page 23.]

[BLJ04] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004. [Cited on page 42.]

[BM03] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 3 2003. [Cited on page 41.]

[BSR+05] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM. [Cited on page 51.]

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. [Cited on page 113.]

[CD09a] K. Cui and Y. Du. Application of boolean kernel function svm in face recognition. In *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 1, pages 619–622, 2009. [Cited on page 39.]

[CD09b] K. Cui and Y. Du. Short-term load forecasting based on the bkf-svm. In *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 2, pages 528–531, 2009. [Cited on page 39.]

[CHW08] K. Cui, F. Han, and P. Wang. Research on face recognition based on boolean kernel svm. In *2008 Fourth International Conference on Natural Computation*, volume 2, pages 148–152, 2008. [Cited on page 39.]

[CK16] Evangelia Christakopoulou and George Karypis. Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 67–74, New York, NY, USA, 2016. ACM. [Cited on page 50.]

[CKM13] Corinna Cortes, Marius Kloft, and Mehryar Mohri. Learning kernels using local rademacher complexity. In *Advances in neural information processing systems*, pages 2760–2768, 2013. [Cited on page 42.]

[CKS+03] Kai-Min Chung, Wei-Chun Kao, Chia-Liang Sun, Li-Lun Wang, and Chih-Jen Lin. Radius margin bounds for support vector machines with the RBF kernel. *Neural Computation*, 15(11):2643–2681, 2003. [Cited on page 114.]

[CMEC17] Rose Catherine, Kathryn Mazaitis, Maxine Eskénazi, and William W. Cohen. Explainable entity-based recommendations with knowledge graphs. In *Proceedings of the Poster Track of the 11th ACM Conference on Recommender Systems (RecSys 2017), Como, Italy, August 28, 2017.*, 2017. [Cited on page 6.]

[CMR10] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Generalization bounds for learning kernels. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 247–254, 2010. [Cited on page 42.]

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995. [Cited on page 28.]

[CVBM02] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine learning*, 46(1):131–159, 2002. [Cited on page 44.]

[DA16] Michele Donini and Fabio Aiolli. Learning deep kernels in the space of dot product polynomials. *Machine Learning*, pages 1–25, 2016. [Cited on pages 105, 106, 109, and 132.]

[DHR+17] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE*

Transaction on Pattern Analysis and Machine Intelligence, 39(4):677–691, 2017. [Cited on page 4.]

[Die08] Joachim Diederich. *Rule Extraction from Support Vector Machines: An Introduction*, pages 3–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. [Cited on page 5.]

[DK04] Mukund Deshpande and George Karypis. Item-based top-*n* recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004. [Cited on page 50.]

[DK11] Christian Desrosiers and George Karypis. *A Comprehensive Survey of Neighborhood-based Recommendation Methods*, pages 107–144. Springer US, Boston, MA, 2011. [Cited on page 48.]

[DK13] Huyen Do and Alexandros Kalousis. Convex formulations of radius-margin based support vector machines. In *International Conference on Machine Learning*, pages 169–177, 2013. [Cited on page 45.]

[DKWH09] Huyen Do, Alexandros Kalousis, Adam Woznica, and Melanie Hilario. Margin and radius based multiple kernel learning. *Machine Learning and Knowledge Discovery in Databases*, pages 330–343, 2009. [Cited on pages 5, 44, and 115.]

[Don16] Michele Donini. *Exploiting the structure of feature spaces in kernel learning*. PhD thesis, University of Padova, 2016. [Cited on page 41.]

[EL06] T. A. Etchells and P. J. G. Lisboa. Orthogonal search-based rule extraction (osre) for trained neural networks: a practical and efficient approach. *IEEE Transactions on Neural Networks*, 17(2):374–384, 2006. [Cited on page 4.]

[EMB+09] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *AISTATS*, volume 5 of *JMLR Proceedings*, pages 153–160. JMLR.org, 2009. [Cited on page 4.]

[Faw06] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006. [Cited on page 17.]

[FSR05] Glenn Fung, Sathyakama Sandilya, and R. Bharat Rao. Rule extraction from linear support vector machines. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 32–40. ACM, 2005. [Cited on page 5.]

[GA11] Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12(Jul):2211–2268, 2011. [Cited on pages 5 and 42.]

[GCZ10] Kun Gai, Guangyun Chen, and Chang-shui Zhang. Learning kernels with radiuses of minimum enclosing balls. In *Advances in neural information processing systems*, pages 649–657, 2010. [Cited on page 44.]

[GF16] Bryce Goodman and Seth Flaxman. Eu regulations on algorithmic decision-making and a "right to explanation", 2016. presented at 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016), New York, NY. [Cited on page 6.]

[GM09] Yann Guermeur and Emmanuel Monfrini. Radius-margin bound on the leave-one-out error of the llw-m-svm. In *ASMDA 2009: XIII International Conference Applied Stochastic Models and Data Analysis*, pages 1–6, 2009. [Cited on page 44.]

[GPH17] Christina Göpfert, Lukas Pfannschmidt, and Barbara Hammer. Feature Relevance Bounds for Linear Classification. In *Proceedings of the ESANN*. 25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2017. [Cited on page 86.]

[GR03] Ashutosh Garg and Dan Roth. Margin distribution and learning algorithms. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 210–217. AAAI Press, 2003. [Cited on page 33.]

[GRGP01] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001. [Cited on page 23.]

[GSK+17] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017. [Cited on page 4.]

[GT10] João Guerreiro and Duarte Trigueiros. *A Unified Approach to the Extraction of Rules from Artificial Neural Networks and Support Vector Machines*, pages 34–42. Springer Berlin Heidelberg, 2010. [Cited on pages 4 and 5.]

[GZTYS14] G. Guo, J. Zhang, D. Thalmann, and N. Yorke-Smith. Etaf: An extended trust antecedents framework for trust prediction. In *Proceedings of the 2014 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 540–547, 2014. [Cited on page 23.]

[GZYS13] G. Guo, J. Zhang, and N. Yorke-Smith. A novel bayesian similarity measure for recommender systems. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2619–2625, 2013. [Cited on page 23.]

[HH13] David Money Harris and Sarah L. Harris. *Digital Design and Computer Architecture (Second Edition)*. Morgan Kaufmann, Boston, second edition edition, 2013. [Cited on pages 19 and 89.]

[HK15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):19:1–19:19, 2015. [Cited on page 23.]

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008. [Cited on pages 48, 49, and 139.]

[HL02] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Trans. Neur. Netw.*, 13(2):415–425, 2002. [Cited on page 21.]

[HM82] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, apr 1982. [Cited on page 17.]

[HS05] Tâm Huynh and Bernt Schiele. Analyzing features for activity recognition. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies*, sOc-EUSAI '05, pages 159–163, New York, NY, USA, 2005. ACM. [Cited on page 4.]

[HSS08] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning, 2008. [Cited on pages 4 and 32.]

[HVPD17] R. Heckel, M. Vlachos, T. Parnell, and C. Duenner. Scalable and interpretable product recommendations via overlapping co-clustering. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1033–1044, 2017. [Cited on page 6.]

[HWL+15] Shanshan Huang, Shuaiqiang Wang, Tie-Yan Liu, Jun Ma, Zhumin Chen, and Jari Veijalainen. Listwise collaborative filtering. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 343–352, New York, NY, USA, 2015. ACM. [Cited on page 51.]

[KCT+01] Lukasz A. Kurgan, Krzysztof J. Cios, Ryszard Tadeusiewicz, Marek Ogiela, and Lucy S. Goodenday. Knowledge discovery approach to automated cardiac spect diagnosis. *Artif. Intell. Med.*, 23(2):149–169, October 2001. [Cited on page 21.]

[KK01] John F. Kolen and Stefan C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243. Wiley-IEEE Press, 2001. [Cited on page 4.]

[KK12] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In Neil D. Lawrence and Mark A. Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, volume 22, pages 583–591, 2012. [Cited on page 132.]

[KP02] V. Koltchinskii and D. Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *The Annals of Statistics*, 30(1):1–50, 02 2002. [Cited on page 41.]

[KRS01] Roni Khardon, Dan Roth, and Rocco Servedio. Efficiency versus convergence of boolean kernels for on-line learning algorithms. In *Proceedings of the 14th In-*

ternational Conference on Neural Information Processing Systems: Natural and Synthetic, NIPS'01, pages 423–430. MIT Press, 2001. [Cited on page 37.]

[KS05] Roni Khardon and Rocco A. Servedio. Maximum margin algorithms with boolean kernels. *Journal of Machine Learning Research*, 6:1405–1429, 2005. [Cited on page 61.]

[KT14] Y. Kusunoki and T. Tanino. Boolean kernels and clustering with pairwise constraints. In *2014 IEEE International Conference on Granular Computing (GrC)*, pages 141–146, 2014. [Cited on page 38.]

[KW71] G. S. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971. [Cited on page 32.]

[LBLL09] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, June 2009. [Cited on page 4.]

[LC09] Shugang Liu and Kebin Cui. Applications of support vector machine based on boolean kernel to spam filtering. *Modern Applied Science*, 3(10), sep 2009. [Cited on page 39.]

[LDA17] Ivano Lauriola, Michele Donini, and Fabio Aiolli. Learning dot-product polynomials for multiclass problems. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2017. [Cited on page 106.]

[LSST+02] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002. [Cited on page 29.]

[LYL14] Xinwang Liu, Jianping Yin, and Jun Long. On radius-incorporated multiple kernel learning. In *Modeling Decisions for Artificial Intelligence*, pages 227–240. Springer, 2014. [Cited on pages 43 and 45.]

[MBMEL12] Brian McFee, Thierry Bertin-Mahieux, Daniel P.W. Ellis, and Gert R.G. Lanckriet. The million song dataset challenge. In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 909–916, New York, NY, USA, 2012. ACM. [Cited on pages 23 and 50.]

[MC80] R. S. Michalski and C. Chilausky. Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, 4:125––161, 1980. [Cited on page 21.]

[Mer09] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446, 1909. [Cited on page 28.]

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. [Cited on pages 3 and 25.]

[MR89] Christopher J. Matheus and Larry A. Rendell. Constructive induction on decision trees. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, pages 645–650, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. [Cited on page 21.]

[NK11] Xia Ning and George Karypis. SLIM: sparse linear methods for top-n recommender systems. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 497–506, 2011. [Cited on pages 49, 50, and 139.]

[NN14] Sinh Hoa Nguyen and Hung Son Nguyen. *Applications of Boolean Kernels in Rough Sets*, pages 65–76. Springer International Publishing, 2014. [Cited on page 37.]

[NPnBT07] Roland Nilsson, José M. Peña, Johan Björkegren, and Jesper Tegnér. Consistent feature selection for pattern recognition in polynomial time. *Journal of Machine Learning Research*, 8:589–612, 2007. [Cited on page 86.]

[NTS90] Michiel O. Noordewier, Geoffrey G. Towell, and Jude W. Shavlik. Training knowledge-based neural networks to recognize genes in dna sequences. In *Proceedings of the 1990 Conference on Advances in Neural Information Processing Systems 3*, NIPS-3, pages 530–536, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc. [Cited on page 21.]

[OAR16] L. Oneto, D. Anguita, and S. Ridella. A local Vapnik-Chervonenkis complexity. *Neural Networks*, 82:62–75, Oct 2016. [Cited on page 41.]

[ODNDSM13] Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 85–92, New York, NY, USA, 2013. ACM. [Cited on page 51.]

[OGRA15] L. Oneto, A. Ghio, S. Ridella, and D. Anguita. Local Rademacher Complexity: sharper risk bounds with and without unlabeled samples. *Neural Networks*, 65:115–125, May 2015. [Cited on page 41.]

[PA16] Mirko Polato and Fabio Aiolli. A preliminary study on a recommender system for the job recommendation challenge. In *Proceedings of the Recommender Systems Challenge*, RecSys Challenge '16, pages 1:1–1:4, New York, NY, USA, 2016. ACM. [Cited on page 48.]

[PC07] Pearl Pu and Li Chen. Trust-inspiring explanation interfaces for recommender systems. *Knowledge-Based Systems*, 20(6):542 – 556, 2007. [Cited on page 6.]

[Pla98] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998. [Cited on page 121.]

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013. [Cited on page 4.]

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. [Cited on page 79.]

[PWCG01] Paul Pavlidis, Jason Weston, Jinsong Cai, and William Noble Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, RECOMB '01, pages 249–255, New York, NY, USA, 2001. ACM. [Cited on page 43.]

[QL09] Shibin Qiu and Terran Lane. A framework for multiple kernel support vector regression and its applications to sirna efficacy prediction. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 6(2):190–199, April 2009. [Cited on page 43.]

[RDGF16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. [Cited on page 4.]

[RF14] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 273–282. ACM, 2014. [Cited on page 51.]

[RFGST09] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461. AUAI Press, 2009. [Cited on pages 49 and 139.]

[RSSB05] Liva Ralaivola, Sanjay J. Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093 – 1110, 2005. [Cited on page 39.]

[Sad01] Ken Sadohara. *Learning of Boolean Functions Using Support Vector Machines*, pages 106–118. Springer Berlin Heidelberg, 2001. [Cited on pages 36, 61, and 68.]

[Sad02] Ken Sadohara. On a capacity control using boolean kernels for the learning of boolean functions. In *Proceedings of the 2002 IEEE International Conference on*

Data Mining, ICDM '02, pages 410–417. IEEE Computer Society, 2002. [Cited on pages 37 and 39.]

[Sch42] I. J. Schoenberg. Positive definite functions on spheres. *Duke Mathematical Journal*, 9(1):96–108, 1942. [Cited on page 105.]

[Sch87] Jeffrey Curtis Schlimmer. *Concept Acquisition Through Representational Adjustment*. PhD thesis, 1987. [Cited on page 21.]

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. [Cited on page 4.]

[Sha87] Alen D. Shapiro. *Structured Induction in Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987. [Cited on page 21.]

[SLH10] Yue Shi, Martha Larson, and Alan Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 269–272, New York, NY, USA, 2010. ACM. [Cited on page 51.]

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013. [Cited on page 4.]

[SRSS06] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7(Jul):1531–1565, 2006. [Cited on page 42.]

[SS01] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. [Cited on pages 4, 25, 29, 57, and 132.]

[StC99] John Shawe-taylor and Nello Cristianini. Further results on the margin distribution. In *In Proc. 12th Annu. Conf. on Comput. Learning Theory*, pages 278–285. ACM Press, 1999. [Cited on pages 33 and 43.]

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. [Cited on pages 4, 29, 38, 88, and 115.]

[STV04] B. Schölkopf, K. Tsuda, and J-P. Vert. *Kernel Methods in Computational Biology*. Computational Molecular Biology. MIT Press, Cambridge, MA, USA, 2004. [Cited on pages 4 and 29.]

[SYZ15] Wanhua Su, Yan Yuan, and Mu Zhu. A relationship between the average precision and the area under the roc curve. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ICTIR '15, pages 349–352, New York, NY, USA, 2015. ACM. [Cited on page 18.]

[TBB+91] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van De Welde, W. Wenzel, J. Wnek, and J. Zhang. The monk's problems a performance comparison of different learning algorithms. Technical report, 1991. [Cited on pages 19 and 21.]

[TBCH11] Shulong Tan, Jiajun Bu, Chun Chen, and Xiaofei He. *Using Rich Social Media Information for Music Recommendation via Hypergraph Model*, pages 213–237. Springer London, 2011. [Cited on page 48.]

[Tin07] Nava Tintarev. Explanations of recommendations. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, pages 203–206, New York, NY, USA, 2007. ACM. [Cited on page 6.]

[TN04] Koji Tsuda and William Stafford Noble. Learning kernels from biological networks by maximizing entropy. *Bioinformatics*, 20(suppl 1):326–333, 2004. [Cited on page 42.]

[TSN90] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 2*, AAAI'90, pages 861–866. AAAI Press, 1990. [Cited on pages 19 and 21.]

[Tsu00] H. Tsukimoto. Extracting rules from trained neural networks. *IEEE Transactions on Neural Networks*, 11(2):377–389, 2000. [Cited on page 4.]

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. [Cited on pages 31, 33, and 44.]

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. [Cited on pages 41 and 44.]

[VC00] V. Vapnik and O. Chapelle. Bounds on error expectation for support vector machines. *Neural Computation*, 12(9):2013–2036, Sep 2000. [Cited on page 44.]

[VSKB10] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010. [Cited on page 29.]

[Wat99] C. Watkins. Kernels from matching operations. Technical report, CSD-TR 98-07, University of London, Computer Science Department, Royal Holloway, 1999. [Cited on page 37.]

[Wol96] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computing*, 8(7):1341–1390, October 1996. [Cited on page 31.]

[WWB+13] Beidou Wang, Can Wang, Jiajun Bu, Chun Chen, Wei Vivian Zhang, Deng Cai, and Xiaofei He. Whom to mention: Expand the diffusion of tweets by recommendation on micro-blogging systems. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1331–1340, New York, NY, USA, 2013. ACM. [Cited on pages 48 and 51.]

[ZLC05] Yang Zhang, Zhanhuai Li, and Kebin Cui. *DRC-BK: Mining Classification Rules by Using Boolean Kernels*, pages 214–222. Springer Berlin Heidelberg, 2005. [Cited on page 38.]

[ZLKY03] Yang Zhang, Zhanhuai Li, Muning Kang, and JianFeng Yan. Improving the classification performance of boolean kernels by applying occam's razor. 2003. [Cited on pages 37 and 39.]

[ZLMJ16] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. *DeepRED – Rule Extraction from Deep Neural Networks*, pages 457–473. Springer International Publishing, 2016. [Cited on page 4.]

[ZMKL05] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM. [Cited on page 23.]

[ZPX+14] Hao Zhong, Weike Pan, Congfu Xu, Zhi Yin, and Zhong Ming. Adaptive pairwise preference learning for collaborative recommendation with implicit feedbacks. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 1999–2002, New York, NY, USA, 2014. ACM. [Cited on page 51.]

[ZSJC05] Ying Zhang, HongYe Su, Tao Jia, and Jian Chu. *Rule Extraction from Trained Support Vector Machines*, pages 61–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. [Cited on page 5.]

[ZTTY+07] Cao Zhe, Qin Tao, Liu Tie-Yan, Tsai Ming-Feng, and Li Hang. Learning to rank: From pairwise approach to listwise approach. Technical report, April 2007. [Cited on page 51.]